Report to
CONFERENCE on DATA
SYSTEMS LANGUAGES

Including
INITIAL SPECIFICATIONS
for a COMMON BUSINESS
ORIENTED LANGUAGE (COBOL)
for Programming
Electronic Digital Computers

DEPARTMENT OF DEFENSE   APRIL 1960

OFFICE OF THE ASSISTANT SECRETARY OF DEFENSE
WASHINGTON 25, D. C.

COMPTROLLER

This report has been prepared through a cooperative effort of computer users in industry, the Department of Defense and other Federal Government agencies and computer manufacturers. It is believed that the use of Common Business Oriented Language (COBOL) can assist materially through reduced programming efforts, in achieving a more effective and economical utilization of electronic digital computers.

The material is published for instructional and informational purposes. Its use in Department of Defense agencies is encouraged.

Comments and suggestions pertaining to this publication should be addressed to the Director, Data Systems Research Staff, Office of the Assistant Secretary of Defense (Comptroller), Washington 25, D. C.

W. Carl Blaisdell
Deputy Comptroller
Financial and Operating Management

COBOL

Initial Specifications

for a

COmmon Business Oriented Language

This report was prepared by the Short Range Task Force of the Conference on Data Systems Languages and accepted by the Executive Committee January 7-8, 1960.

This report is a technical reference manual detailing the initial specifications of COBOL. It is not intended to be a training or teaching manual, and assumes a fair knowledge of data processing on the part of the reader.

FOREWORD

To the Conference on Data Systems Languages

Subject; COBOL - Initial Specifications for a COmmon Business Oriented Language

At a meeting January 7 - 8, 1960, the Executive Committee accepted and approved for publication and distribution to the Conference the subject report of the Short Range Committee dated December 17, 1959. The Executive Committee believes that such initial specifications for COBOL are a major contribution in the development of a single business data processing language. COBOL represents the only method of expressing business data processing problems acceptable by such a wide group of data processing systems. Most of the manufacturers of data processing equipment have recognized the benefits to all users and to manufacturers of using a common programming language and most of the manufacturers have agreed to provide COBOL compilers as part of their programming service to customers.

In addition to editing the report (and preliminary specifications) for typographical and other minor errors, the Executive Committee rewrote Part I, Introduction. Section 4, Phasing, and Section 5, Maintenance, now reflect the Executive Committee's wish to emphasize the fact that deficiencies in the initial specifications are well recognized together with the establishment of a mechanism by which such deficiencies can be overcome promptly and effectively.

The Conference on Data Systems Languages is a voluntary cooperative effort of users of data processing systems (both in the government and industry) and manufacturers of data processing systems. The objective of this effort is to develop a common language, basically in English, which is oriented toward business data processing problems, open-ended and independent of any make or model of data processing equipment. The initial specifications for such a COmmon Business Oriented Language (COBOL) as set forth herein represent the first milestone toward this objective.

The Executive Committee recommends that users of general purpose computers consider the use of COBOL in programming business data processing problems.

Chairman
Executive Committee
Conference on Data Systems Languages

TABLE OF CONTENTS

iv

DETAILED OUTLINE OF COBOL REPORT

VI. DATA DIVISION

I - INTRODUCTION

1. OBJECTIVES

There are hundreds of business, government, and educational organizations using a wide variety of electronic computers in data processing operations. Some of the major users have more than one type of computer applied to the same general data processing application at different locations. The experience of these organizations to date indicates that a major problem in using computing equipment wisely and efficiently lies in stating the data processing application in such a way that computer programs are developed and maintained with a minimum of time and programming effort.

A COmmon Business Oriented Language, independent of any make or model of computer, open-ended and stated in English, would do much to solve or reduce this problem. Such a language would also simplify and speed up the related problem of training personnel in the design of data processing systems and the development of computer programs for such systems.

In general, the following situations represent some of the areas requiring the development of a COmmon Business Oriented Language for programming computers:

a. The need to develop data processing systems for existing computers that can be processed on future, more powerful computers with a minimum of conversion costs. For any user possessing computers of different manufacture and therefore not compatible, this need exists for efficient translation of a data system from one computer type to another.

b. With the rapidly changing and expanding requirements of management, data processing systems need constant revision and augmentation. Full documentation of the present system is required in a form conducive to compilation of such changes and additions with minimum time and costs.

c. Many users are faced with the need to produce a large number of computer programs in a short period of time. This places a heavy burden on the existing programming staff or requires quick augmentation with relatively inexperienced programmers.

2. HISTORY

On May 28 and 29, 1959, a meeting was called in the Pentagon for the purpose of considering both the desirability and the feasibility of establishing a common language for the programming of electronic computers in business-type data processing. Representatives from users, Government installations, computer manufacturers and other interested parties were present. There was almost unanimous agreement that the project was both desirable and feasible at this time. The concept of three committees or task forces was agreed upon. (The terms "committee," "task force" and "group" are used interchangeably in this report.) They were called the SHORT RANGE, INTERMEDIATE RANGE, and LONG RANGE with appropriate time scales. The Short Range Group was composed of six manufacturers and three Government representatives. This Committee held its first meeting

on June 23, 1959. At that time it was decided that the tasks of the Committee fell in four general areas, and working groups were established as follows:

>Data Description
>Procedural Statements
>Application Survey
>Usage and Experience

The first two groups held frequent meetings and prepared proposals for consideration by the full committee which met August 18 - 21 and August 24 - 25 for the purpose of preparing a report to the Executive Committee. Materials developed as the result of the work of the latter two groups were used in the course of the development of COBOL. The report to the Executive Committee, submitted in September 1959, stated that the Short Range Committee felt it had prepared a framework upon which an effective common business language could be built. It was recognized that the report contained rough spots and needed additions. It further requested that the Short Range Committee be authorized to complete and polish the system by December 1959. It was also requested and approved that the Short Range Committee continue beyond December in order to monitor the implementation. Both these requests were approved.

The Committee held meetings between September 18 and November 7, 1959, and proceeded steadily in its task of resolving problems and completing the language. The name "COBOL", which suggests a COmmon Business Oriented Language, was adopted.

The COBOL System was reviewed and approved by the Short Range Committee during the week of November 16 - 20. Final editing and initial distribution of the report to the Executive Committee was accomplished December 17, 1959.

3. ATTRIBUTION

One or more persons from the following companies or Government agencies have participated in the work of the Short Range Committee at one time or another:

>Air Materiel Command, U. S. Air Force
>Bureau of Standards, Department of Commerce
>Computer Science Corporation
>Datamatic Division, Minneapolis-Honeywell Corporation
>David Taylor Model Basin, Bureau of Ships, U. S. Navy
>ElectroData Division, Burroughs Corporation
>International Business Machines Corporation
>Radio Corporation of America
>Remington-Rand Division of Sperry-Rand, Inc.
>Sylvania Electric Products, Inc.

Ideas and information were drawn from many sources; in particular, from the FLOW-MATIC* System developed by Sperry-Rand, the Commercial Translator System designed by IBM, and the AIMACO System developed jointly by the Air Materiel Command and Sperry-Rand. With the permission of the authors and

*Trademark of Sperry-Rand Corproation

publishers, certain material has been taken from the following copyrighted publications: FLOW-MATIC* Programming System, © 1958 Sperry-Rand Corporation, and General Information Manual: IBM Commercial Translator © 1959 by International Business Machines Corporation.

4. PHASING

In the development of any system there are problems of time and values which must be solved by the designers. In the development of the COBOL system, consideration was given to the varying amounts of time needed for implementation, the time needed by the Committee to determine certain specifications, and the importance (i.e., value) of these specifications. Since opinions on these elements vary, the Short Range Committee proposed a concept called "phasing". The Committee further defined three phases of development:

> The minimum or Phase I - Basic COBOL
> The present level or Phase II - COBOL
> A more ideal system or Phase III - Extended COBOL

While agreeing in general with the concept and the need for recognizing different levels of implementation, the Executive Committee considers it unrealistic to attempt to establish precise boundaries for such levels or phases in this initial manual. The concept of phasing is not intended to place a limitation upon the system as a data processor but rather to recognize the limitations that may be necessary as to time and state of compatibility of the language. All compilers which accept only COBOL terms are proper COBOL processors, however, it must be recognized that complete compatibility will not be achieved until all manufacturers have implemented all features contained in the COBOL language.

In developing the specifications contained herein, the Short Range Committee went beyond what they consider to be the minimum language (referred to in the original report as Basic COBOL, Phase 1) which was specifically defined as:

1. Only those features in the ENVIRONMENT DIVISION which permit the practical operation of the I-O system according to the requirements of particular implementors.

2. All features of the DATA DIVISION

3. All features and verbs in the PROCEDURE DIVISION except

   a. Algebraic formulas and the COMPUTE verb
   b. Segmentation
   c. Conditional sentences containing more than one IF
   d. REVERSE option in the OPEN verb
   e. VARY option in the PERFORM verb with respect to the use of BY, FROM, TO "field-name"
   f. UNTIL option in the PERFORM verb
   g. INCLUDE verb
   h. USE verb

*Trademark of Sperry-Rand Corporation

i. DEFINE verb
j. ENTER verb
k. CORRESPONDING option in the MOVE verb

The Executive Committee agrees with this definition as representing, in a general sense, the minimum language specifications that a manufacturer would be expected to implement through a compiler, keeping in mind that no manufacturer would be expected to implement aspects or features that are not normal data processing practice for their type of computer.

The Executive Committee does not believe that it would be realistic at this time to attempt to define (as "Extended COBOL") or to identify the additional features that may be added to these specifications. The Short Range Committee identified and listed in the original report additional features which should be added as rapidly as possible, such as:

> Table Handling Functions
> External Format and Media Translation
> Report Writing Extension
> Sort-Merge Functions

The prospective user of COBOL is reminded that a source program language such as COBOL must be dynamic to be effective and will be subject to continuous change. Computer manufacturers should advise prospective users very specifically as to what features of COBOL are provided for in compilers at any particular point in time.

5. MAINTENANCE

It is clear that the task of defining a COmmon Business Oriented Language does not end with the publication of preliminary specifications. Continuing attention to the system is needed in order to answer questions arising from users and implementors of the language and to make definitive modifications (including additions, clarifications, and changes).

The Executive Committee did not accept the method of maintaining COBOL which was recommended by the Short Range Task Force in the December 17, 1959, report. Instead, a subcommittee was formed to give this very important subject careful consideration and present recommendations thereon. Such recommendations were presented to and adopted by the Executive Committee at the February 12, 1960, meeting and are outlined hereafter.

Two committees were established and charged jointly with responsibilities for COBOL maintenance. These committees are:

Technical Committee

> Membership of the Technical Committee shall consist of one representative of each of the six manufacturers of electronic computers who participated in the Short Range Committee plus any other manufacturer interested in actively participating in the maintenance of COBOL. Manufacturers not members of the original group will apply for membership to the Executive Committee explaining their interest. The member representing each

manufacturer should be the top person in automatic programming techniques, or represent such person, and be in a position to speak for the company.

The Technical Committee shall consider, from the manufacturers' standpoint, all proposals to supplement COBOL - reviewing them for need, technical feasibility and practicality - and shall approve or disapprove as a Committee action all such proposals. Proposals generated within the Technical Committee will be sent to the Executive Committee for assignment of a proposal number and referral to the Maintenance Committee for concurrent consideration.

Maintenance Committee

Membership of the Maintenance Committee shall initially consist, in addition to the Chairman, of one representative of the Air Force, the Navy, U. S. Steel and Esso Standard Oil, together with a cross-section of other major users of data processing systems in Government or industry interested in actively participating in the maintenance of COBOL. Users not members of the original group will apply for membership to the Executive Committee explaining their interest. The member representing each user should be the top person in the use of automatic programming techniques, or represent such person, and be in a position to speak for the company or agency.

The Maintenance Committee shall consider, from the users' standpoint, all proposals to supplement COBOL for need, technical feasibility and practicality and shall approve or disapprove as a Committee action all such proposals. Proposals generated within the Maintenance Committee will be sent to the Executive Committee for assignment of a proposal number and referral to the Technical Committee for concurrent consideration.

All additions, clarifications and changes to COBOL will be reproduced and released by the Executive Committee as a number "supplement." Proposals for supplements to COBOL will be received from outside organizations and individuals by the Executive Committee, will be assigned a proposal number and sent to the Technical and Maintenance Committees for concurrent consideration. Proposals generated within the Technical or Maintenance Committees will also be sent to the Executive Committee for assignment of a proposal number. Approval by both the Technical and Maintenance Committees will result in the immediate adoption and distribution of an official numbered supplement to COBOL. In case of a split vote, the Executive Committee will resolve the difference. Periodically, these supplements will be combined and published in the same manner as the original COBOL. When a proposal has been approved by Committee action of either the Technical or Maintenance Committee and referred for action of the other Committee, the latter group must take Committee action within two weeks. Failure to take Committee action in the two week period will be considered as approval.

Recognizing the lack of a clear-cut distinction between the intermediate range and the long-range efforts, together with the need for a clarification of such activities as they relate to COBOL and its maintenance, the planning subcommittee submitted further recommendations which were adopted by the Executive Committee February 12, 1960. Such recommendations provided for the consolidation of the

long and intermediate-range efforts in a Development Committee, retaining the present membership of the Intermediate-Range Task Force within such Committee. The membership and responsibilities of the Development Committee were defined as follows:

> Membership of the Development Committee shall consist of representatives from industry, government, universities, consultants, and manufacturers of data processing systems and others with competence and interest in the furtherance of programming language development as a means of advancing the applications of ADPS.

> . The Committee will conduct or review research in the field of programming which will permit use by all systems analysts regardless of their professions or subject-matter specialty, to describe the process to be performed in such a manner as to be meaningful and appropriate for any concept of implementation. Proposals for Development Committee projects can originate in the Committee, the Executive Committee or from outside sources and will be assigned a project number by the Executive Committee. The Committee will make periodic reports on the status of such projects. Proposals for supplements to COBOL may generate from Development Committee projects.

A schematic diagram (Figure 1) is included to show the Committee relationships as they relate to COBOL maintenance.

6. ACKNOWLEDGMENT

It is requested of all organizations who intend to implement the COBOL system and expect to write a manual describing the operation of their processor of the COBOL system that the remainder of the Acknowledgment Section be included in its entirety as part of the preface to any publication.

"This publication is based on the COBOL System developed in 1959 by a voluntary committee composed of government users and computer manufacturers. The organizations participating in the original development were:

> Air Materiel Command, U. S. Air Force
> Bureau of Standards, Department of Commerce
> Datamatic Division, Minneapolis-Honeywell Corporation
> David Taylor Model Basin, Bureau of Ships, U. S. Navy
> ElectroData Division, Burroughs Corporation
> International Business Machines Corporation
> Radio Corporation of America
> Remington-Rand Division of Sperry-Rand, Inc.
> Sylvania Electric Products, Inc.

"These initial specifications for the COBOL language are the result of contributions made by all of the above-mentioned organizations and no warranty expressed or implied, as to the accuracy and functioning of the programming system and language is made by any contributor or by the committee and no responsibility is assumed by any contributor or by the committee in connection therewith.

"It is reasonable to expect that many improvements and additions will be made to COBOL. Every effort will be made to insure that improvements and corrections

will be made in an orderly fashion making proper provision not to invalidate existing users' investments in programming. However, this can be positively assured only by individual implementors.

"Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures and the methods for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

"The authors and copyright holders of the copyrighted material used herein: FLOW-MATIC (Trade-mark of Sperry Rand Corporation) Programming for the UNIVAC ® I and II, Data Automation Systems © 1958, 1959, Sperry Rand Corporation; IBM Commercial Translator, Form No. F 28-8013, copyrighted 1959 by IBM, have specifically authorized the use of this material, in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

"Any organization interested in reproducing the COBOL report and initial specifications in whole or in part, using ideas taken from this report or utilizing this report as the basis for an instruction manual or any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention "COBOL" in acknowledgment of the source but need not quote the entire section."

CONFERENCE ON DATA SYSTEMS LANGUAGES
ORGANIZATION FOR COBOL MAINTENANCE

Figure 1

ORGANIZATIONS

Corporotions, etc.
A.C.M.

USER GROUPS
Share, Use,
Guide, et ol

Individuols

Manufacturers

University and other technical & computer interested groups

Ideas Comment Criticism Proposals etc

DEVELOPMENT COMMITTEE

Conducts research pro-jects ossigned or opproved by Exec. Comte. involving mojor change or improve-ment in the common pro-gromming longuage. Re-ports to Exec. Comte. on project stotus. Completed projects will normally become proposols for supplements to COBOL.

Projects

Reports

Proposols

EXECUTIVE COMMITTEE

Assigns & maintains control record on Projects referred to Development Committee & Proposals for COBOL supple-ments referred to Technical & Maintenance Committees. Provides for immediate publi-cation of proposals accepted by both Committees. Resolves split votes between Technical Comte & Maintenance Comte on proposals.

Unnumbered Proposols

Numbered Proposols

*Comte. action

TECHNICAL COMMITTEE

Considers all numbered proposols referred by Exec-utive Committee from stond-point of the computer manu-focturers and takes committee oction thereon. May also generate proposals.

Unnumbered Proposols

Numbered Proposols

*Comte. oction

MAINTENANCE COMMITTEE

Considers all numbered proposals referred by Exec-utive Committee from stand-point of the computer users ond takes committee oction thereon. Moy olso generate proposols.

Publicotion of COBOL supplements

*When one committee has acted fovorobly on a proposol, the other committee must take committee action within two weeks or assent is ossumed.

II. GENERAL DESCRIPTION OF COBOL

1. GENERAL PHILOSOPHY OF COBOL DEVELOPMENT

The task of the committee was that of preparing a common business language. By this is meant the establishment of a standard method of expressing solutions for a certain class of problems normally referred to as business data processing. The word "Common" was interpreted to mean that the source program language would be compatible among computers. Differences in computers relating to size, types of peripheral equipment and different order structure make this impossible in its entirety. Thus, the goal of achieving the maximum amount of compatibility on present day computers was the realistic framework within which all work was done.

In describing a data processing problem, there are two elements involved. One is the set of procedures which specify how the data is to be manipulated and the other is a description of the data involved. Furthermore, it was recognized that certain information pertaining to the computer itself was a necessary part of the description of a problem. This, of course, would never carry over from one computer to another. However, it was felt that the advantages of a common means of expression were sufficiently great to develop a standard form for even those items which clearly changed from computer to computer.

As this philosophy developed, COBOL was crystallized into three parts. They are listed in decreasing order of compatibility among computers.

> Procedural Statements
> Data Description
> Environment Description

2. COBOL SYSTEM DESCRIPTION

The COBOL System is composed of two elements – the source program written in COBOL, and the compiler which translates this source program into an object program capable of running on a computer. This report considers only the source program and does not consider the second element directly, although the specifications of a language obviously determine to a large extent the boundaries of a compiler. However, the compiler is mentioned in certain cases to facilitate explanation of the language.

A source program is used to specify the solution of a business data processing problem. The three elements of this specification are:

1. The set of procedures which determine how the data is to be processed.

2. The description of the data being processed.

3. The description of the equipment being used in the processing.

The COBOL System has a separate division within the source program for each. The names of these divisions are: PROCEDURE, DATA, and ENVIRONMENT.

The COBOL System allows the user to prepare his specifications for the problem solution in the language most natural to him – namely English.

The PROCEDURE DIVISION specifies the steps that the user wishes the computer to follow. These steps are expressed in terms of meaningful English words, statements, sentences, and paragraphs. This aspect of the overall system is often referred to as the "program"; in reality it is only part of the total specification of the problem solution (i.e. the program) and is insufficient to describe the entire problem. This is true because repeated references must be made – either explicitly or implicitly – to information appearing in the other divisions. This division – more than any other – allows the user to express his thoughts in meaningful English. Concepts of verbs to denote actions, and sentences to describe procedures, are basic, as is the use of conditional clauses to provide alternative paths of action. The PROCEDURE DIVISION is essentially computer independent. That is, any user of COBOL can understand the information appearing in this division without regard to any particular computer. Furthermore, every COBOL compiler will interpret this information in the same way.

The DATA DIVISION uses file and record descriptions to describe the files of data that the object program is to manipulate or create, and the individual logical records which comprise these files. Most physical characteristics of the files are not included here, so that this division is to a certain extent computer independent. While compatibility among computers cannot in general be assured, careful planning in the data layout will permit the same data descriptions to apply to more than one computer.

The ENVIRONMENT DIVISION is that part of the source program which specifies the equipment being used. It contains descriptions of the computers to be used both for compiling the source program and running the object program. Memory size, number of tape units, hardware switches, printers, etc. are among many items that may be mentioned for a particular computer. Problem oriented names also may be assigned to particular equipment. Those aspects of a file which relate directly to hardware are also described here. Because this division deals entirely with the specifications of the equipment being used, it is largely computer dependent.

The amount of inter-computer compatibility throughout the COBOL System varies with the division and the effort expended to obtain this goal. In the PROCEDURE DIVISION, virtually no effort is needed to maintain compatibility among computers. In the DATA DIVISION, care must be taken to minimize the loss of object program efficiency. In the ENVIRONMENT DIVISION, almost all information is computer dependent and therefore the compatibility is based on ease of understanding rather than direct transference.

The COBOL System is the first large scale effort in defining a single language which permits the writing of data processing problems for many computers. In addition to the compatibility which has been achieved, the COBOL System provides the user with an effective means of describing the solution of his data processing problems.

III. CHARACTERS AND WORDS

1. CHARACTER SET

1.1 CHARACTERS USED FOR WORDS

The character set for words will consist of the 37 characters

0, 1, ... , 9
A, B, ... , Z
-                        (hyphen or minus)

Note particularly, that "blank" or "space" is not an allowable character for a word, but is used to separate words and statements. Where a "blank" or "space" is employed, more than one may be used, except in the Reference Formats. See VIII. Groups of characters selected from the 37 characters are called "words".

1.2 CHARACTERS USED FOR PUNCTUATION

The punctuation characters consist of the following:

"	Quotation Mark
(	Left Parenthesis
)	Right Parenthesis
	Space
.	Period
,	Comma
;	Semicolon

1.3 CHARACTERS USED IN FORMULAS

+	Addition
- (hyphen)	Subtraction
*	Multiplication (**Exponentiation)
/	Division
=	Equality

1.4 CHARACTERS USED IN RELATIONS

>	Greater Than
<	Less Than
=	Equal to

1.5 CHARACTERS USED IN EDITING

$	Dollar Sign
*	Check Protection Symbol
,	Comma
.	Actual Decimal Point

1.6 SUMMARY OF ALLOWABLE CHARACTERS

Those characters which are recognizable by the COBOL system include the letters of the alphabet, decimal integers, the punctuation characters, and those characters commonly called symbols, which are used in formulas, relations and editing.

Because all computers may not have the complete list of characters defined above, single character substitutions may be made as required. When fewer than 51 characters are available, double characters may be substituted for the single characters defined. A standard list of these substitutions will be specified at a later date.

It is also conceivable that other characters, which appear within a particular computer's character set, may be desirable for use within literals or constants. The possibility exists, however, that the use of characters other than those previously defined as the proper COBOL set may result in a loss of compatibility depending upon the particular computer used.

2. WORDS

2.1 DEFINITION OF WORDS

A word is composed of not more than 30 characters chosen from the following set of 37:

> 0 - 9
> A - Z
> hyphen (i.e., minus)

A word is ended by a space, or a word is ended by either a period, right parenthesis, comma, semicolon, followed by a space. A word may not begin or end with a hyphen. A literal constitutes an exception to the above rules. See Literals 2.2.1d.

The use of punctuation characters in connection with words is as follows: A period, comma, and semicolon when used, must always immediately follow a word, but they must be followed by a space. A left parenthesis or a beginning quote mark (see Literals) must not be followed by a space unless the space is desired in the literal. A right parenthesis or ending quote mark must not be preceded by a space unless the space is desired in the literal.

2.2 TYPES OF WORDS

2.2.1 Nouns

A noun is a single word which is one of the following:

> Data Name
> Condition Name
> Procedure Name
> Literal
> Figurative Constant
> Special Register Name
> Special Names

A noun may contain hyphens for readability purposes. For example,

quantity-on-hand
stock-number

are legitimate nouns, whereas,

stocknumber-

is not. (Labels, tags, field names, operation numbers, and other such terms used in other languages are considered nouns in this language.)

a) Data-Names

A data-name is a word with at least one alphabetic character, which designates any data specified in the data description. (File names and field names refer to two specific data levels).

b) Condition-Names

A condition-name is the name assigned to a value which a field may assume. A condition-name must contain at least one alphabetic character. The field itself is called a conditional variable, and those values which it may assume are referred to by condition-names. The actual value of the condition-name is defined in the Record Description.

For example, consider a conditional variable called TITLE. If the condition-names ANALYST, PROGRAMMER, and CODER are assigned the values 1, 2, and 3, respectively, the conditional expression:

IF CODER THEN . . . . .

would generate a test of the field TITLE against the value "3".

Condition-names may also be defined in the SPECIAL-NAMES paragraph within the ENVIRONMENT DIVISION. Here, a condition-name is given to the on and/or off status of hardware devices. For example, the device SENSE FLIP-FLOP may be called a conditional variable. The two values the variable can assume may be named PRESENT and ABSENT. Thus, the conditional expressions:

IF PRESENT THEN . . . . . or

IF ABSENT THEN . . . . .

may be used. The major difference in this type of condition-name is that the definition of the actual value of the condition-names is automatically handled by the compiler.

A condition-name, then, is a name assigned to a value which a field may have at a given time. This field, called a conditional variable, may have many different values and a condition-name for each value. The same condition-name may not be used for more than one value of the same variable. Furthermore, condition-names may only be used in conditional expressions.

c) Procedure-Names

A procedure, either a paragraph or a section, i.e. a group of paragraphs, may be named to permit one procedure to refer to others. The procedure-name may be purely numeric. Only those procedures which are referred to within the program need be named.

Procedure-names are applied either to paragraphs or to sections and are accordingly known as paragraph names or section names.

d) Literals

A literal is a noun whose value is identical to those characters comprising the noun. The literal may be either numeric, alphabetic, or alphanumeric and may be any length. If a literal is alphabetic or alphanumeric, it must be bounded by quotation marks, and may contain any allowable character except the quotation marks. Three consecutive quote marks constitute a valid literal consisting of a single quote mark.

A numeric literal is defined as a group of characters chosen from the numerals 0 through 9, a plus (+) or minus (-) sign and a decimal point (.), and may - but need not - be bounded by quotation marks. A numeric literal not bounded by quotation marks may not be terminated by a decimal point.

The computer's internal representation of a literal will be determined by the compiler from context. See each verb for the rules governing the use of literals in the PROCEDURE DIVISION.

e) Figurative Constants

Certain literals, called figurative constants, have been assigned fixed names. These names, when used as figurative constants, must not be bounded by quotation marks.

These names are:

ZERO	HIGH-VALUE
ZEROES	HIGH-VALUES
ZEROS	LOW-VALUE
SPACE	LOW-VALUES
SPACES	ALL any literal

Figurative constants generate a string of homogeneous information whose length is dependent upon context. For example, MOVE ALL 4 FILLING FIELD, where FIELD is defined as six characters in length would result in 444444. Whereas, MOVE ALL, "FOUR" FILLING FIELD, would result in FOURFO.

f) Special Register

TALLY is the name of a special register whose length is equivalent to a five decimal digit field. Its primary use is to hold information produced by the EXAMINE verb. It may also be used, however, to hold information produced

elsewhere in a program. The compiler will allocate memory for the TALLY field only if reference is made to it.

g) Special Names

Special names provide a means of relating hardware with problem-oriented names and the status of hardware switches with condition-names. See Special Names VII 2.3.

2.2.2 Verbs

A verb is a single word which appears in the PROCEDURE DIVISION and designates an action.

2.2.3 Reserved Words

The reserved words are used for syntactical purposes and may not be used as nouns or verbs. There are three types:

a) Connectives

Connectives are words used to denote the presence of a qualifier or the presence of a subscript.

Logical connectives are used to indicate the appearance of an independent clause and also to aid in defining the precise rules for the evaluation of compound conditionals.

b) Optional Words

Optional words have been defined to improve the readability of the language. The presence or absence of these optional words does not alter the compiler's interpretation of the statement.

c) Key Words

In some formats, certain words are required to complete the meaning of the verbs or entries and therefore these words must be present.

2.3 SPECIAL USAGE

2.3.1 Qualifiers

Every name in a COBOL program must be unique, either because no other name has the identical spelling or because the name exists within a hierarchy of names such that the name can be made unique by mentioning several higher elements in the hierarchy. The higher elements are called qualifiers when used in this way, and the process is called qualification. With each use of a name, enough qualification must be mentioned to make the use unambiguous, but is not necessary to mention all possible levels of qualification unless they are needed for uniqueness. A file-name is the highest level qualifier available for a data-name. A section-name is the highest level qualifier available for a paragraph-name.

Two types of qualification are allowed: prefixing (i.e., adjectival modification) and suffixing. In qualifying a single occurrence of a data-name, only one type may be used. In the first type, the nouns must appear in descending order of hierarchy, i.e., with the name being qualified as the last, and all others in order. In the second type, the nouns must appear in ascending order of hierarchy with either of the words OF or IN separating them. The choice between IN or OF is based only on readability because they are logically equivalent.

If, for example, two records, MASTER and NEW-MASTER, each contains a CURRENT-DATE field and a LAST-TRANSACTION-DATE field, and if each of these fields contains three subfields, MONTH, DAY, and YEAR, the current month in the NEW-MASTER record may be referred to as:

MONTH IN CURRENT-DATE OF NEW-MASTER

NEW-MASTER CURRENT-DATE MONTH

; while the day of the last transaction in the Master record may be referred to as:

DAY IN LAST-TRANSACTION-DATE OF MASTER

MASTER LAST-TRANSACTION-DATE DAY

The following rules are used for Qualifications:

1. A qualifier must exist outside (above) the name it is qualifying.

2. A name may not appear at two levels in a hierarchy so that it would appear to qualify itself.

3. If a data-name or condition-name appears more than once in the DATA DIVISION of a program, it must be qualified in all references occurring in the PROCEDURE DIVISION. The absence of qualification may not be considered qualification.

4. A paragraph-name must not be duplicated within the same Section. There is an exception to this rule which is discussed in Chapter IX 1. LIBRARIES. A paragraph-name may be qualified by a section-name. When it is, the word SECTION must not appear. A paragraph-name need not be qualified when referred to from within the same Section.

2.3.2 Subscripts

The technique of subscripting is most commonly used for table handling functions. A subscript is an integer whose value determines which element within a table (or a list) of like elements of data is to be operated upon. The integer may be represented by a literal or by a data-name.

The name of the element being subscripted is followed by its subscript. The subscript, itself, is identified either by preceding it with the key word FOR, or by surrounding it with parentheses ().

A subscript value of "1" denotes the first element of a list, a value of "2", the second element, etc. No element within a table may be referenced without a subscript; however, the entire table may be referenced, provided the table has been given a unique name.

> Examples: MOVE rate FOR age TO listing.
> IF height (10) IS GREATER THAN -----
> MULTIPLY price FOR 5 BY inventory -----
> EXAMINE class (region) REPLACING -----
> MOVE rate-table TO storage-area.

Tables are often defined such that more than one level of subscripting is required to locate an element within the table. A maximum of three dimensions, or levels, is permitted by COBOL. Multi-level subscripts are always written from left to right: major, intermediate, and minor.

Reference to a data-name within a table must include all subscripts upon which the data-name is dependent. Use of more or less than the correct number of subscripts is considered an error. For example, the premium rate of an insurance policy might depend upon the age, the weight, and the state of residence. A table of such rates would be considered as three dimensional. Therefore, reference to any rate within the table would always be followed by three subscripts. Multi-level subscripts must be enclosed in parentheses.

> Examples: MULTIPLY policy-value BY rate (age, weight, state).
> SUBTRACT rate (10, 5, 7) FROM -----

2.3.3 Series

Several related nouns may be written as a series separated by any one of the following:

> ,
> , AND
> , OR
> AND
> OR

The comma must be adjacent to the preceding word, and followed by a space. The words AND and OR are considered key words.

> Example: OPEN INPUT MASTER, RECEIPTS, AND ORDERS
> ADD ABLE AND BAKER AND CHARLIE

IV. NOTATION USED IN VERB AND ENTRY FORMATS IN THIS REPORT

1. All upper case words which are underlined are required, and an error will occur if they are absent or incorrectly spelled.

2. All upper case words which are not underlined are used for readability only. They may be present or absent.

3. All lower case words represent generic quantities whose value must be supplied by the user.

4. Material in braces $\{ \ \}$ indicates that a choice from the contents must be made.

5. Material in square brackets $[\ \ ]$ represents an option and may be included or omitted at the user's choice.

6. Notes will elaborate and specify any restrictions.

7. In cases where many choices were available, some separations into numbered options have been made.

8. In series of two or more nouns, they have been shown separated by commas. However, connectives

,

AND

, AND

are equivalent and may be used interchangeably.

V. PROCEDURE DIVISION

1. GENERAL DESCRIPTION

The PROCEDURE DIVISION contains those operations needed to solve a given problem. (This is commonly called the program, but is only one of the three parts of the program.) These operations are formally divided into SECTIONS each of which is subdivided into PARAGRAPHS. Only sections and paragraphs can be named, and therefore they are the only elements to which control may be transferred.

2. FORMULAS

A formula consists of nouns representing quantities upon which arithmetic may be performed, and arithmetic operators, combined according to the rules of algebraic expressions. (SEE APPENDIX A). These rules assume the sequence for executing the operations to be exponentiation, then multiplication and division, and finally addition and subtraction.

There are six arithmetic operators which may be used in formulas. They may be expressed by the character representing the operator, in which case it must be surrounded by spaces, or their English equivalent may be used. The following choices are available:·

	=	EQUALS
(Addition)	+	PLUS
(Subtraction)	-	MINUS
(Multiplication)	*	MULTIPLIED BY or TIMES
(Division)	/	DIVIDED BY
(Exponentiation)	**	EXPONENTIATED BY

3. CONDITIONALS.

3.1 GENERAL DESCRIPTION

Conditional procedures are one of the keystones in describing data processing problems. COBOL makes available to the programmer several means of expressing conditional situations.

COBOL conditionals generally involve the key word IF followed by the conditions to be examined followed by the operations to be performed. Depending on the truth or falsity of the conditions different sets of operations are to be performed.

Several different types of sentences involving more than one IF are available.

3.2 CONDITIONS

3.2.1 General Definition

A condition is a group of words which can be determined to be either true or false, i.e., whose truth value can be determined.

3.2.2 Simple Condition

A simple condition consists of either a condition name, a relation or a test.

a) Condition Name

A field whose specific values can be named is called a conditional variable. The names given to the values are called Condition-Names. These may be tested to determine whether or not the designated value is present. Thus the truth or falsity may be established.

b) Relations

The relations available in COBOL are shown below.

$$
\underline{IF} \ \left[\begin{Bmatrix} \text{field} \\ \text{literal} \\ \text{formula} \end{Bmatrix}\right] \ \begin{Bmatrix} \text{IS} \ [\underline{NOT}] \ \underline{GREATER} \ \underline{THAN} \\ \text{IS} \ [\underline{NOT}] \ \underline{LESS} \ \underline{THAN} \\ \text{IS} \ [\underline{NOT}] \ \underline{EQUAL} \ \underline{TO} \\ \text{IS} \ \underline{UNEQUAL} \ \underline{TO} \\ \underline{EQUALS} \\ \underline{EXCEEDS} \end{Bmatrix} \ \left[\begin{Bmatrix} \text{field} \\ \text{literal} \\ \text{formula} \end{Bmatrix}\right]
$$

The following named operators and equivalent symbols may be used interchangeably if the latter are available in the character set of the computer:

GREATER THAN	>
LESS THAN	<
EQUAL TO	=

The left hand term of the relation is called the subject and the right hand term is called the object. If the subject or the object or both do not appear, the term(s) of the last executed relation in the same sentence is used for the missing one(s). A relation missing either term, therefore, must not be used as the first relation to be tested within a sentence. When comparing alphabetic or alphanumeric quantities, the adjustments necessitated by different collating sequences will be handled automatically by the compiler.

The benefit gained by allowing the naming of the subject and/or the object of a relation to be optional can best be shown by the following example:

IF X EQUALS Y, MOVE A TO B; IF GREATER MOVE A TO C; IF LESS MOVE A TO D.

c) Tests.

It is possible to determine the status of a field by means of the following tests:

$$\underline{IF} \quad [field] \quad IS \quad \begin{Bmatrix} [\underline{NOT}] & \underline{NUMERIC} \\ [\underline{NOT}] & \underline{POSITIVE} \\ [\underline{NOT}] & \underline{NEGATIVE} \\ [\underline{NOT}] & \underline{ZERO} \end{Bmatrix}$$

If a field is not named, then the test is made on the result of the last arithmetic operation executed in the same sentence. A field must be shown when NUMERIC is used.

The explicit interpretations of these terms is as follows: A field is NUMERIC if it consists of the digits zero to nine with or without a sign. A field is POSITIVE only if it is greater than zero. A field whose value is zero is NOT POSITIVE. A field is NEGATIVE only if it is less than zero. A field whose value is zero is NOT NEGATIVE. More briefly, the value zero is never considered positive or negative.

3.2.3 Compound Condition

A compound condition is a sequence of simple conditions connected by either of the words AND, OR. It is not necessary to use the same word as a connective throughout, however, if both AND and OR appear, the rules of evaluation given below must be carefully followed.

The most general form of a compound condition may be expressed symbolically as:

$$\text{Simple-Cond-1} \begin{Bmatrix} \text{AND} \\ \underline{OR} \end{Bmatrix} \text{Simple-Cond-2} \begin{Bmatrix} \text{AND} \\ \underline{OR} \end{Bmatrix} \dots \text{Simple-Cond-n}$$

This is evaluated by the rules given in APPENDIX B.

It is understood that although any simple condition (i.e., condition name, relation, test) may be compounded, only the "relation" contains both a subject and an object. COBOL allows the abbreviation of compound conditional "relations" so that an implied subject for both multiple relations and series of objects can be permitted.

552133 O - 60 - 3

The abbreviated forms, each of which has its own specific meaning, are the following:

Form 1:

$$IF \left\{ \begin{array}{l} \text{field-1} \\ \text{literal-1} \\ \text{formula-1} \end{array} \right\} \left(\text{Relation} \right) \left\{ \begin{array}{l} \text{field-2} \\ \text{literal-2} \\ \text{formula-2} \end{array} \right\} \left\{ \begin{array}{l} \underline{AND} \\ OR \end{array} \right\} \left(\text{Relation} \right) \left\{ \begin{array}{l} \text{field-3} \\ \text{literal-3} \\ \text{formula-3} \end{array} \right\}$$

$$\left\{ \begin{array}{l} \underline{AND} \\ OR \end{array} \right\} \left(\text{Relation} \right) \ldots \left\{ \begin{array}{l} \underline{AND} \\ OR \end{array} \right\} \left(\text{Relation} \right) \left\{ \begin{array}{l} \text{field-n} \\ \text{literal-n} \\ \text{formula-n} \end{array} \right\}$$

This is the same as if

$$\left\{ \begin{array}{l} \text{field-1} \\ \text{literal-1} \\ \text{formula-1} \end{array} \right\}$$
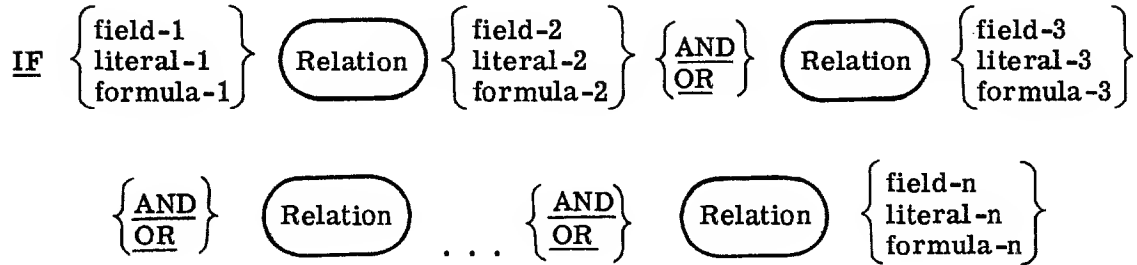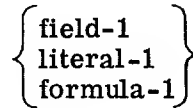
appeared immediately preceding every

$$\left(\text{Relation} \right)$$

Example:

A = B OR C AND D

is the same as

A = B OR A = C AND A = D

which is defined in APPENDIX B to be

A = B OR (A = C AND A = D)

Form 2:

$$IF \left\{ \begin{array}{l} \text{field-1} \\ \text{literal-1} \\ \text{formula-1} \end{array} \right\} \left(\text{Relation} \right) \left\{ \begin{array}{l} \text{field-2} \\ \text{literal-2} \\ \text{formula-2} \end{array} \right\} ,$$

$$\left\{ \begin{array}{l} \text{field-3} \\ \text{literal-3} \\ \text{formula-3} \end{array} \right\} , \ldots \left\{ \begin{array}{l} \underline{AND} \\ OR \end{array} \right\} \left\{ \begin{array}{l} \text{field-n} \\ \text{literal-n} \\ \text{formula-n} \end{array} \right\}$$
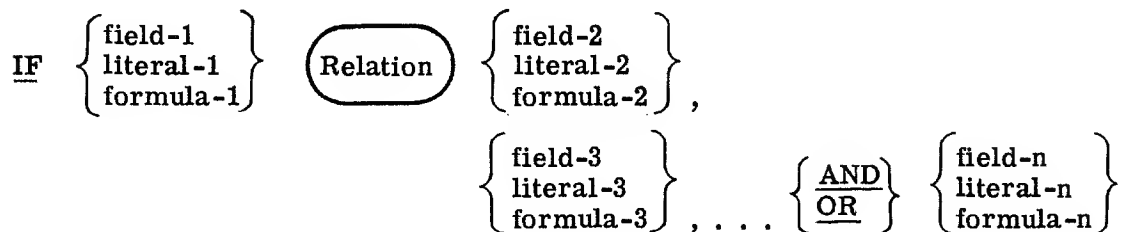
This is equivalent to Form 1 with all commas replaced by whichever one of the words AND or OR was used.

Example:

$$X = 2, \ Y \ OR \ Z$$

is equivalent to

$$X = 2 \ OR \ X = Y \ OR \ X = Z$$

4. RULES OF FORMATION OF PROCEDURES

4.1 GENERAL APPROACH

COBOL procedures are expressed in a manner similar (but not identical) to normal English prose. The largest unit is a section, which is composed of paragraphs. These are made up of sentences which are generally grouped for the purpose of describing a unified idea. The sentences are composed of sequences of statements, which in turn are made up of groups of words - normally verbs and operands.

4.2 EXPRESSIONS

4.2.1 Imperative Expression

An imperative expression consists of a verb and its operands.

4.2.2 Conditional Expression

A conditional expression consists of a condition preceded by the word IF and followed by one of the following:

THEN

a comma (,) followed by a space and then a verb

a verb

4.3 STATEMENTS

4.3.1 Imperative Statement

An imperative statement is a single imperative expression (i.e., a verb and its operands) which is terminated by one of the following:

AND ALSO

;   (semicolon)

The verbs GO and STOP RUN may not be followed by other expressions. No compiler directing verb (see Section 6) may be preceded or followed by any expression.

4.3.2 Conditional Statement

A conditional statement consists of a conditional expression followed by either

a) any imperative statement, or a sequence of imperative statements

or   b) any sequence of imperative statements, the last of which is terminated by a semicolon and followed by the word <u>OTHERWISE</u> which is then followed by any sequence of imperative statements, the last of which may be followed by any conditional statement.

Example: IF X EQUALS Y, MOVE A TO B;

IF X EQUALS Z THEN MOVE A TO B; OTHERWISE ADD A TO C AND ALSO IF NOT POSITIVE, GO TO ERROR-ROUTINE; OTHER-WISE ADD A TO B;

4.4 SENTENCES

4.4.1 Imperative Sentence

An imperative sentence consists of a sequence of one or more imperative statements, the last of which terminates with a period.

a) Simple Imperative Sentence

A simple imperative sentence consists of an imperative statement terminated by a period.

Example: MOVE A TO B.

b) Compound Imperative Sentence

A compound imperative sentence consists of a sequence of more than one imperative statement, the last of which is terminated by a period.

Example: MOVE A TO B; ADD C TO D.

4.4.2 Conditional Sentence

A conditional sentence consists of a sequence of one or more conditional statements such that the rules for connecting them shown below are satisfied and the last statement terminates with a period.

a) Simple Conditional Sentence

A simple conditional sentence consists of one conditional statement which is either terminated by a period or followed by any imperative sentence.

Examples: IF X EQUALS Y THEN MOVE A TO B.

IF X EQUALS Y THEN MOVE A TO B; ADD C TO D.

IF X EQUALS Y, MOVE A TO B; OTHERWISE IF C EQUALS D, MOVE A TO D AND ALSO PERFORM PATH-1 THRU PATH-6.

b) Compound Conditional Sentence

A compound conditional sentence is a sequence of conditional statements such that the last and only the last is terminated by a period and all others are terminated by one of the following:

;   (semicolon)

AND ALSO

AND

OR

The same terminator must be repeated throughout the sentence for all statements except that the phrase AND ALSO and the punctuation mark semicolon (;) have identical logical significance. For purposes of separating conditional statements, AND is logically equivalent to AND IF and OR is logically equivalent to OR IF because regardless of any other rules, the expression IF IF is never meaningful.

For the exact rules concerning the evaluation of compound conditional sentences, see Section 5 of this chapter.

Examples: IF X EQUALS Y THEN MOVE A TO B; IF Y EQUALS Z THEN ADD B TO C.

IF X EQUALS Y THEN MOVE A TO B; OTHERWISE IF C EQUALS D THEN MOVE A TO D AND IF Y EQUALS Z THEN ADD B TO C.

IF X EQUALS Y MOVE A TO B; IF GREATER MOVE A TO C; OTHERWISE MOVE A TO D.

4.5 PARAGRAPHS

Paragraphs permit the grouping of several sentences to convey one idea. It is not necessary to name a paragraph unless it is to be ALTERed or entered out of sequence by the program. When a paragraph name is given, it applies until either a new paragraph name appears or indentation occurs, which indicates the start of an unnamed paragraph. (See Chapter VIII. 2, REFERENCE FORMAT).

4.6 SECTIONS

A section when designated must be named and consists of one or more paragraphs. The section-name is followed by the word SECTION and a period. The section-name applies to all paragraphs following it until another section-name

is encountered. A program is not required to be separated into sections. For further details see REFERENCE FORMAT, VIII-3.

5. EVALUATION OF CONDITIONAL SENTENCES

5.1 GENERAL NOTATION

There are three types of conditional sentences which provide great power and scope in the handling of logical situations. The rules for evaluating these sentences can most easily be shown by means of flow charts. It should be understood that these flow charts are simply a means of expressing the concepts involved and do not represent the method by which the implementation will be done.

The following notation is used:

C1 and C2 represent conditions (simple or compound).

S1 and S2 represent imperative statements.

W represents a sentence (imperative or conditional).

NS represents the next sentence.

T represents the truth of the condition.

F represents the falsity of the condition.

5.2 SIMPLE CONDITIONAL EVALUATION

A simple conditional sentence is defined as a conditional statement which is either terminated by a period or followed by an imperative sentence. The general forms of this may be written as :

IF C1 THEN S1.

IF C1 THEN S1; OTHERWISE W.

IF C1, S1.

IF C1, S1; OTHERWISE W.

where C1 is a condition, and S1 and W are sequences of imperative statements.

Another way of writing the most general form is:

IF condition THEN imp-statem-1; imp-statem-2; ... ; imp-statem-n;

OTHERWISE imp-statem-nl; imp-statem-n2; ... ; imp-statem-nn.

If the condition (C1) is true, then statement-1 through statement-n is executed and then control is transferred to the next sentence. If the condition (C1) is false then control is immediately transferred to statement-nl (which is the OTHERWISE path) and this, through statement-nn, are executed. The program

then continues at the next sentence, unless any Si contains a control transfer to a named paragraph.

The interpretation of this sentence may be shown diagrammatically by the following chart:
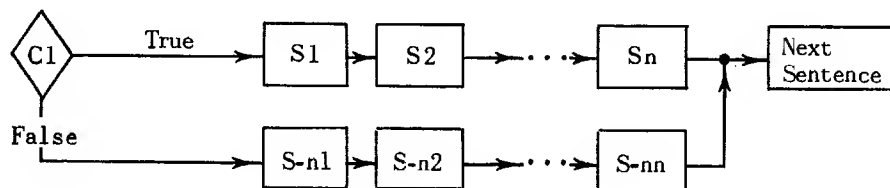
5.3 COMPOUND CONDITIONAL EVALUATION

There are three types of compound conditional sentences whose rules must be specified. Note that the simple form shown in 5.2 is actually a collapsed form of each of the three general cases. If any Si contains a control transfer then it is executed, and the remainder of the flow chart is ignored.

5.3.1 AND IF

The following sentences are logically equivalent and are evaluated according to the flow chart below:

IF C1 THEN S1 AND IF C2 THEN S2; OTHERWISE W.

IF C1, S1 AND IF C2 THEN S2; OTHERWISE W.

IF C1 THEN S1 AND IF C2, S2; OTHERWISE W.
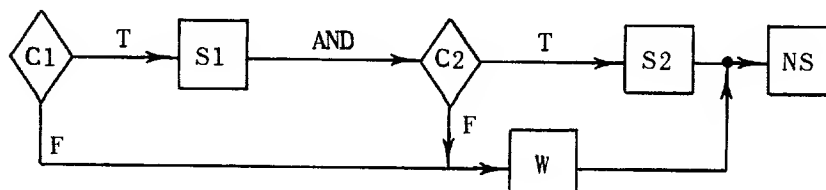
5.3.2 OR IF (Inclusive)

The following sentences are logically equivalent and are evaluated according to either of the flow charts below. They are simply two different methods of representing the same idea.

IF C1 THEN S1 OR IF C2 THEN S2; OTHERWISE W.

IF C1, S1 OR IF C2 THEN S2; OTHERWISE W.

IF C1 THEN S1 OR IF C2, S2; OTHERWISE W.
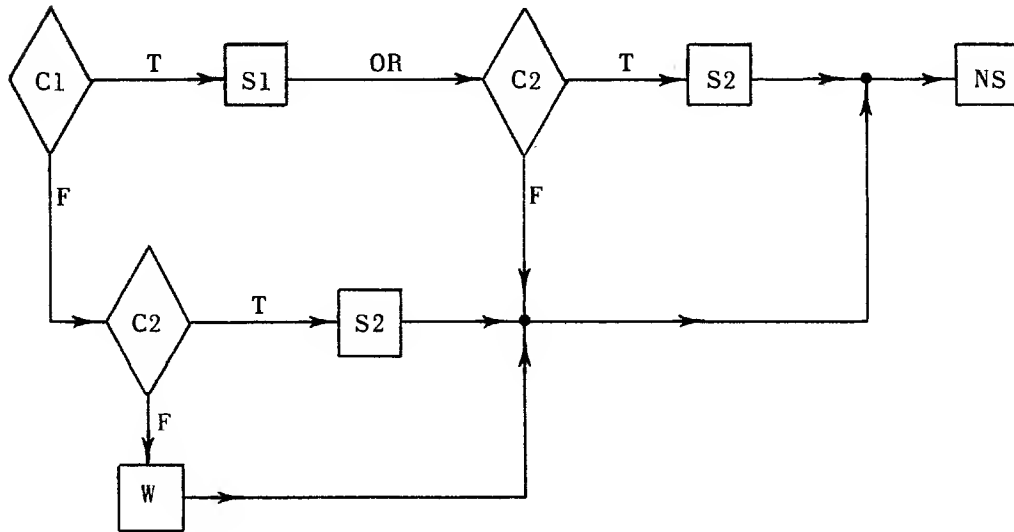
(In the following, SW represents a switch.)
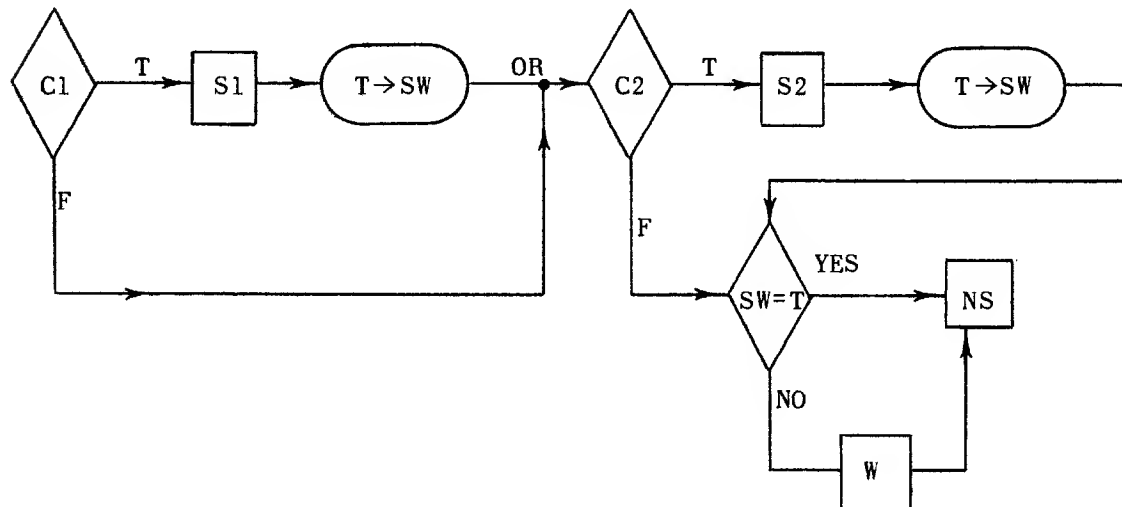
5.3.3 AND ALSO IF or (; IF)

The following sentences are logically equivalent and are evaluated according to either of the flow charts below. They are simply two different methods of representing the same idea.

IF C1 THEN S1 AND ALSO IF C2 THEN S2; OTHERWISE W.

IF C1, S1 AND ALSO IF C2 THEN S2; OTHERWISE W.

IF C1 THEN S1 AND ALSO IF C2, S2; OTHERWISE W.
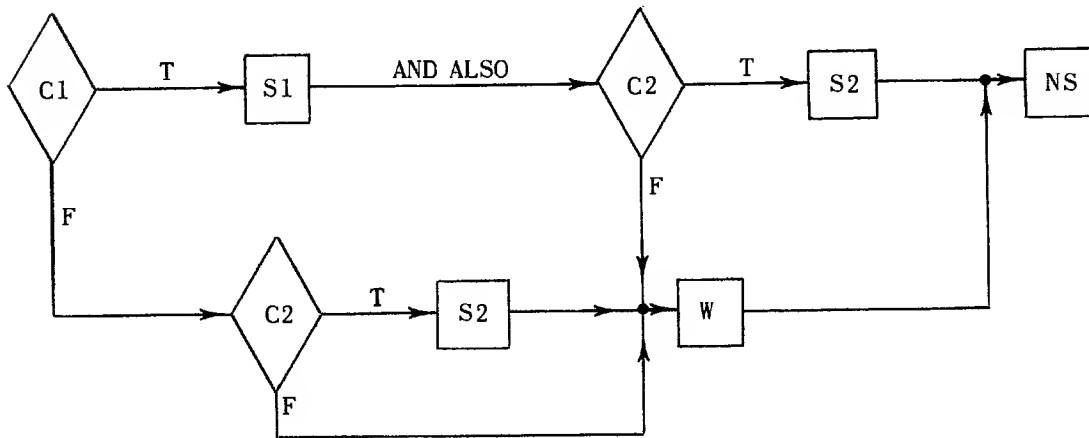
(In the following, SW represents a switch.)
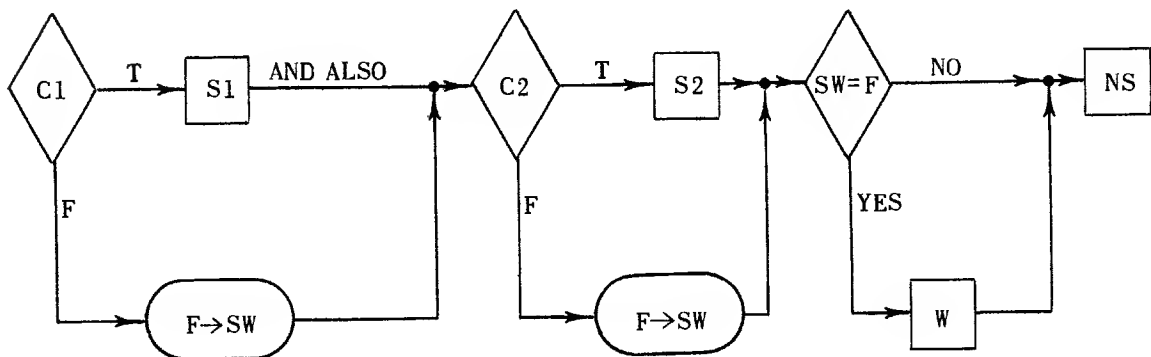
6. <u>VERBS</u>

6.1 LIST BY CATEGORIES

Arithmetic	ADD SUBTRACT MULTIPLY DIVIDE COMPUTE
Input-Output	READ WRITE OPEN CLOSE ACCEPT DISPLAY
Procedure Branching	GO ALTER PERFORM
Data Movement	MOVE EXAMINE
Ending	STOP
Compiler Directing Verbs	DEFINE ENTER EXIT INCLUDE NOTE USE

<u>Note</u>: Although the word IF is not a verb in the strictest sense, it possesses one of the most important characteristics of one - namely the generation of coding in the object program. Its occurrence is a vital feature in the PROCEDURE DIVISION, and is fully discussed in Sections 4 and 5 of this Chapter.

6.2 FORMATS

The verb formats are shown on the following pages.

ACCEPT

FUNCTION: To allow a unit of low volume input from available devices such as card readers, paper tape readers, console keyboards, etc.

ACCEPT data-name FROM mnemonic-name

Notes:

1. The "mnemonic-name" corresponds to a particular "hardware" unit defined in the Special Names paragraph in the ENVIRONMENT DIVISION.

FUNCTION: To add two or more quantities and store the sum in either the last named field or the specified one.

$$\underline{\text{ADD}} \left\{ \begin{array}{l} \text{literal-1} \\ \text{field-name-1} \end{array} \right\} , \left\{ \begin{array}{l} \text{literal-2} \\ \text{field-name-2} \end{array} \right\} \left[, \left\{ \begin{array}{l} \text{literal-3} \\ \text{field-name-3} \end{array} \right\} \ldots \right]$$

$$\left[\underline{\text{GIVING}} \text{ field-name-n} \right] \left[\underline{\text{UNROUNDED}} \right] \left[; \text{ON } \underline{\text{SIZE}} \ \underline{\text{ERROR}} \text{ any impera-} \right.$$

$$\left. \text{tive statement} \right]$$

Notes:

1. If the "GIVING field-name-n" option is not present, then the last named field receives the result. This field must not be a literal.

2. The maximum size of any operand ("literal" or "field-name") is 18 characters. An error will be indicated at compilation time if the format of any operand specifies a number of characters in excess of this. This does not apply to the intermediate result fields, which will be carried out to one more place on the right and one more place on the left than the maximum field size.

3. Only numeric literals may be used. If a sign (+ or -) is included, it must appear as the most significant character of the literal. The sign digit is converted by the compiler to the correct computer representation when necessary. If a decimal point appears in the literal, it is not included in the stored representation of the literal, and is used solely to align the literal with the associated "field-names."

4. The final result will be stored in accordance with the description of the receiving field as specified in the Record Description. Furthermore, all fields may have different formats. Decimal position alignment for operands is automatically supplied according to the type of operation involved. The decimal position of the result will likewise be aligned prior to storage in the result field.

5. Because the number of decimal places in the calculated result will always be at least one digit greater than the number of decimal places in the result field (See Note 2), automatic rounding will occur whenever the UNROUNDED option is not used. The least significant digit of the result field will be increased by 1 whenever the most significant digit of the excess is greater than or equal to 5.

```
┌─────────┐
│  ADD    │
└─────────┘
```

6. When the UNROUNDED option is used, the number of decimal places in the calculated result will be truncated according to the size of the result field.

7. De-editing (i. e., the removal of any editing symbols such as dollar signs, real decimal points, etc.) will not be supplied. Thus, only "field-name-n" may contain any editing symbols. An error will be indicated at compilation time if the format of any operand specifies the presence of such symbols.

8. Whenever the number of integral places in the computed result exceeds that which may be stored in the result field, a SIZE ERROR condition arises. This causes truncation of the most significant digits of the result field prior to storing. This action occurs regardless of whether or not the SIZE ERROR option is used.

9. The SIZE ERROR option may only be used in conjunction with the arithmetics.

ALTER

FUNCTION: To modify a predetermined sequence of operations.

ALTER paragraph-name-1 TO PROCEED TO procedure-name-2

[paragraph-name-3 TO PROCEED TO procedure-name-4 . . .]

Notes:

1. "Paragraph-name-1", "paragraph-name-3" , . . . are names of para-
graphs which each contain a single sentence consisting of only a GO state-
ment as defined under Option 1 of the GO verb.

<p style="text-align: right;">┌──────────┐
│ CLOSE │
└──────────┘</p>

FUNCTION: To terminate the processing of both input and output reels and files, with optional rewind and/or lock.

$$\underline{\text{CLOSE}} \text{ file-name-1} \left[\underline{\text{REEL}}\right] \left[\text{WITH} \left\{\begin{array}{l}\text{LOCK}\\\underline{\text{NO REWIND}}\end{array}\right\}\right] \left[, \text{ file-name-2} \ldots \right]$$

Notes:

1. A "CLOSE file-name" must be executed once and only once for a given file unless the file has been reopened. It will initiate the final closing conventions for this file and release the data area.

2. REEL option: For both input and output files, the specified rerun procedures and next reel processing are instituted. (See RERUN in the I-O CONTROL Paragraph of the ENVIRONMENT DIVISION and the verbs OPEN, READ and WRITE).

   a) When referring to a reel of an input file, the standard end of reel processing is eliminated for only that reel. If a CLOSE REEL is given for the last reel of a file, an error may occur in the object program. The results of this error will be specified by the individual implementor.

   b) When referring to a reel of an output file, the standard end of reel processing takes place.

3. The LOCK option automatically rewinds the file and supplies the appropriate technique for insuring that the file cannot be read or written upon.

4. If the NO REWIND option is used on a tape file, the tape will remain positioned after the ending label.

5. If neither LOCK nor NO REWIND is specified, the tape will be rewound.

6. If the file were specified as OPTIONAL (see the FILE-CONTROL Paragraph of the ENVIRONMENT DIVISION), the standard end of file processing is not performed whenever this file is not present.

FUNCTION: To permit the use of formulas.

$$\underline{\text{COMPUTE}} \quad \text{field-name-1} \quad \left[\underline{\text{UNROUNDED}}\right] \quad \left\{ \begin{array}{c} \underline{\text{FROM}} \\ = \\ \underline{\text{EQUALS}} \end{array} \right\} \quad \text{any operation}$$

$$\left[; \quad \text{ON} \ \underline{\text{SIZE}} \ \underline{\text{ERROR}} \ \text{any imperative statement} \right]$$

Notes:

1. "Any operation" may be a single field or any meaningful combination of fields parenthesized as required, employing the algebraic symbols for the operations of addition, subtraction, division, multiplication and exponentiation or their English equivalent. The precise rules are defined in Appendix A - "COMPREHENSIVE RULES FOR FORMING ALGEBRAIC EXPRESSIONS".

2. "field-name-1" must not be a literal.

3. All rules specified for the simple arithmetics (ADD, SUBTRACT, MULTIPLY, DIVIDE) apply to the COMPUTE verb.

4. The SIZE ERROR option applies to field-name-1 and not to each intermediate result.

5. The words FROM and EQUALS are equivalent to each other and to the symbol " = ".

FUNCTION: To allow the introduction of new verbs.

DEFINE VERB name AS $\left\{ \begin{matrix} \text{procedure-name-1} \\ \text{any sentence} \end{matrix} \left[\underline{\text{THRU}} \text{ procedure-name-2} \right] \right\}$

WITH FORMAT name . . .

Notes:

1. A DEFINE VERB statement may not use another DEFINE within its definition.

2. The DEFINE causes the replacement of the new verb by those COBOL statements used in its definition.

3. Dummy names used in the DEFINE VERB and in the statements (e.g. ABLE, BAKER, CHARLIE, as used in the example below) must be the same. The substitution takes place based on the position of the names and the other words used.

4. When the newly defined verb is used, its format must be exactly the same as the format shown by the DEFINE, except that optional words will be ignored if present.

5. In place of the three dots (. . .), the actual format is shown. Literals may be used in the format.

EXAMPLE 1:

Defining a subroutine

DEFINE VERB COMPUTE-NUMBER AS

FIRST THRU LAST WITH FORMAT COMPUTE-NUMBER

FROM ABLE AND BAKER AND CHARLIE AS NUMBER.

FIRST. ADD ABLE TO BAKER.

MULTIPLY BAKER BY CHARLIE.

LAST. ADD CHARLIE TO 3 GIVING NUMBER.

```
┌──────────────┐
│              │
│   DEFINE     │
│              │
└──────────────┘
```

Using this subroutine

USE-NUMBER. COMPUTE-NUMBER FROM PAY AND TAX AND

TEN-PERCENT AS NUMBER.

Using this subroutine a second time

EXCEPTION-ROUTINE. COMPUTE-NUMBER FROM

OVERTIME, CITY-TAX, THREE-PERCENT AS DUE-TAX-COLLECTOR.

EXAMPLE 2:

Defining a subroutine

DEFINE VERB SPECADD AS MULTIPLY A BY C GIVING C WITH FORMAT

SPECADD A AND B AND C.

Using this subroutine

CALCULATE. SPECADD QUANTITY AND INVENTORY AND TOTAL.

DISPLAY

FUNCTION: To allow for visual display of low volume information.

$$\underline{\text{DISPLAY}} \left\{ \begin{matrix} \text{literal-1} \\ \text{data-name-1} \end{matrix} \right\} \left[, \left\{ \begin{matrix} \text{literal-2} \\ \text{data-name-2. . .} \end{matrix} \right\} \right] \underline{\text{UPON}} \text{ mnemonic-name}$$

Notes:

1. The "mnemonic-name" corresponds to a particular "hardware" unit defined in the ENVIRONMENT DIVISION.

DIVIDE

FUNCTION: To divide one number into another and store the result in the last named field or the specified one.

$$\underline{\text{DIVIDE}} \quad \left\{ \begin{array}{l} \text{literal-1} \\ \text{field-name-1} \end{array} \right\} \quad \underline{\text{INTO}} \quad \left\{ \begin{array}{l} \text{literal-2} \\ \text{field-name-2} \end{array} \right\} \left[\underline{\text{GIVING}}\ \text{field-name-3} \right]$$

$$\left[\underline{\text{UNROUNDED}} \right] \quad \left[;\ \text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ \text{any imperative statement} \right]$$

Notes:

1. All notes specified under the ADD verb apply to the DIVIDE verb.

2. In addition to the above, division by zero constitutes a special type of "size error". Regardless of whether or not the "size error" option has been specified, an attempted division by zero will leave the result field unaltered.

ENTER

FUNCTION: To provide a means of allowing more than one language in the same program.

ENTER language-name $\left[\text{paragraph-name}\right]$

Notes:

1. The language-name may be any language allowable by the particular compiler. Examples are X-1 for Univac, Autocoder for 705, etc.

2. The paragraph-name is the identifier for the particular portion of other language code written.

3. The other languages may be written directly in line where called or at the end of the source program. If they are in line, then they must be followed by ENTER COBOL to return to the main source language.

4. The other language coding is inserted in line in the object program at the place in which it is called, regardless of where it may have been written by the user.

5. The "paragraph-name" allows the other language section to be referenced by other parts of the COBOL program. Thus, if the other language code is written as a closed subroutine, it must be preceded by a GO to statement and it may be executed by PERFORMING the paragraph-name. The normal case is the execution of the other language section directly in line as an open subroutine.

6. Implementors will specify how to write other languages for their compilers.

EXAMINE

FUNCTION: To replace and/or count occurrences of a given character in data.

$$\underline{\text{EXAMINE}} \quad \text{data-name} \left\{ \begin{array}{l} \underline{\text{TALLYING}} \left\{ \begin{array}{l} \underline{\text{ALL}} \\ \underline{\text{LEADING}} \\ \underline{\text{UNTIL FIRST}} \end{array} \right\} \quad \text{literal-1} \\ \qquad\qquad\qquad \left[\underline{\text{REPLACING}} \text{ WITH literal-2} \right] \\ \underline{\text{REPLACING}} \left\{ \begin{array}{l} \underline{\text{ALL}} \\ \underline{\text{LEADING}} \\ [\underline{\text{UNTIL}}] \underline{\text{FIRST}} \end{array} \right\} \text{literal-3 WITH literal-4} \end{array} \right\}$$

Notes:

1. The literal must be a single character.

2. Examining always starts at the left.

3. If the word LEADING appears, then the replacement and/or the tally begins with the first character in the "data-name" and terminates as soon as a character other than "literal-1" is encountered.

4. The tally is placed in a special register called TALLY. Furthermore,

    a. If the ALL option is used, the tally represents the number of occurrences.

    b. If the LEADING option is used, the tally represents the number of occurrences of "literal-1" prior to encountering a character other than "literal-1".

    c. If the UNTIL FIRST option is used, the tally represents the number of other characters encountered before the occurrence of "literal-1".

EXIT

FUNCTION: To furnish an end point for a loop when required.

EXIT.

Notes:

1. EXIT produces no coding in the object program.

2. EXIT must be preceded by a paragraph-name and appear as a single one-word sentence.

3. EXIT is used in conjunction with the PERFORM verb. If no related PERFORM is in process, sequence control will pass through the EXIT point to the next statement.

GO

FUNCTION: To depart from the normal sequence of procedures.

Option 1:

GO TO [procedure-name]

Option 2:

GO TO procedure-name-1, procedure-name-2, [, procedure-name-3 ...]

DEPENDING ON data-name

Notes:

1. In Option 1, if a GO statement is to be ALTERED, then:

   a. The statement must have a paragraph-name.

   b. The paragraph may contain only a single sentence consisting of only a GO statement.

   The paragraph-name is referred to by the ALTER verb in order to modify the sequence of the program. If the "procedure-name" is omitted, the compiler will insert an error stop in the object program; therefore, the GO statement must be referenced by an ALTER statement prior to the first execution of the GO statement.

2. In Option 2, the "data-name" must have a positive integral value. The branch will be to the 1st, 2nd ... , nth "procedure-name" as the value of "data-name" is 1, 2, ... , n. If the value of the "data-name" is zero, or exceeds n (i. e. the number of procedures named) the next statement in normal sequence will be executed.

```
                                                    ┌──────────────┐
                                                    │   INCLUDE    │
                                                    └──────────────┘
```

FUNCTION: To save the programmer effort by automatically incorporating
library subroutines into the source program.

$$\underline{\text{INCLUDE}}\ \text{subroutine-name}\begin{Bmatrix}\underline{\text{HERE}}\\ \underline{\text{AS}}\ \text{paragraph-name}\end{Bmatrix}\left[\underline{\text{REPLACING}}\begin{Bmatrix}\text{subroutine-word-1}\\ \text{data-name-1}\end{Bmatrix}\right.$$

$$\underline{\text{BY}}\begin{Bmatrix}\text{word-1}\\ \text{data-name-2}\end{Bmatrix}\quad \left[\begin{Bmatrix}\text{subroutine-word-2}\\ ,\text{data-name-3}\end{Bmatrix}\ \underline{\text{BY}}\ \begin{Bmatrix}\text{word-2}\\ \text{data-name-4}\end{Bmatrix}\dots\right]\ \Bigg]$$

Notes:

1. The source program is compiled as though the programmer had written
    the subroutine in its entirety without reference to the library, except
    for the use of DEFINE. Any DEFINE in the subroutine does not affect
    the program, and conversely any DEFINE in the program does not affect
    the subroutine. (See also Chapter IX. 1, Libraries).

2. The replacing of words does not alter the material as it exists in the
    library and it may be called for again in the same program with different
    replacements. Normally the programmer will wish to limit his replace-
    ments to data-names and procedure-names.

3. Specified replacement will not occur within part of a literal, but the
    entire literal may be replaced.

4. If the HERE option is used, the subroutine will be inserted in place of
    the INCLUDE statement, and when needed, the paragraph-name
    associated with the INCLUDE statement can be used to qualify paragraph-
    names defined within the library material.

5. If the "AS paragraph-name" option is used, the compiler will determine
    where the subroutine is to be inserted in the object program. This
    "paragraph-name" may not appear as a "paragraph-name" elsewhere
    in the program. In order to execute the subroutine, a "PERFORM
    paragraph-name" must be used. If the same subroutine is included
    more than once, then there must be a different "paragraph-name" for
    each set of replacements.

6. After inclusion in the program, the subroutine name is not available as
    a qualifier, unless it has otherwise been used as a procedure name.

MOVE

FUNCTION: To transfer data, in accordance with the rules of editing, to one or more data fields.

$$\underline{\text{MOVE}} \begin{Bmatrix} \begin{bmatrix} \underline{\text{CORRESPONDING}} \end{bmatrix} \text{data-name-1} & \underline{\text{TO}} \\ \text{literal-1} & \begin{Bmatrix} \underline{\text{FILLING}} \\ \underline{\text{TO}} \end{Bmatrix} \end{Bmatrix} \text{data-name-2} \begin{bmatrix} , \text{ data-name-3} . . . \end{bmatrix}$$

Notes:

1. If neither the FILLING nor the CORRESPONDING option is used, and if any data-name is a higher level than field size, a straight copy into the receiving area will be made providing the formats of both the sending and receiving areas are identical except for names. If the formats differ, the operation is illegal and no data transmission will be attempted. Fields are (or a literal is) moved according to the rules of field editing shown in Note 4. In any case, no editing symbols will be removed by the MOVE.

2. If the CORRESPONDING option is used, corresponding fields are moved, edited, and arranged according to the rules of field editing. Non-corresponding fields in the sending area are not moved and non-corresponding fields in the receiving area are not altered. No editing symbols will be removed by the MOVE. See Note 5 for complete definition of CORRESPONDING fields.

3. If the FILLING option is used, the set of characters (i. e., the literal) is placed in the receiving area starting at the left hand side. This is repeated until the area is filled, at which time truncation occurs, if necessary.

4. The rules for editing specify that fields are moved in conformity with the data description format of the receiving area.

   a) For numeric fields this includes:

      1. Alignment of decimal points with truncation or zero fill on either end as required.

      2. Zero suppression, insertion of dollar signs, commas, decimal points, etc., and justification as specified in the Record Description.

      3. Conversion of type of representation (e. g., numeric or binary mode to alphanumeric mode and in special cases alphanumeric to numeric or binary mode).

```
┌─────────┐
│  MOVE   │
└─────────┘
```

    b) For non-numeric fields this includes:

       1. Left justification unless otherwise specified in the detailed data description.

       2. Space fill if the receiving field is larger than the sending field.

       3. If the receiving field is smaller than the sending field the operation is illegal.

5. In order for fields to be called CORRESPONDING as mentioned in Note 2, the following situations must exist:

    a) There must be two areas larger than field size having names which are different or which can be made different by qualification.

    b) Within these areas there must be pairs of fields which have identical names and which have the same number of levels intervening between the fields and their respective larger areas.

    c) Corresponding fields may contain within them corresponding pairs. If the outer corresponding fields are of the same format, a MOVE of the outer corresponding fields will be generated. If the inner pair differs in size or format the MOVE CORRESPONDING operation will ignore the higher correspondence and will operate upon the lower correspondence. In cases of corresponding at more than two levels, this process will be repeated. It is, of course, impossible for outer fields to have different formats when the inner fields are alike.

6. If the hierarchy is identical in size and format, the outer pair is moved; otherwise successive corresponding pairs are examined.

7. As the COBOL compiler will, upon encountering a MOVE CORRESPONDING option generate a series of MOVES, it follows that the rules for editing specified for MOVE will apply after determination of the specific MOVES which result from each MOVE CORRESPONDING operation.

FUNCTION: To multiply two quantities together and store the result in the last named field or the specified one.

$$\underline{\text{MULTIPLY}} \begin{Bmatrix} \text{literal-1} \\ \text{field-name-1} \end{Bmatrix} \underline{\text{BY}} \begin{Bmatrix} \text{literal-2} \\ \text{field-name-2} \end{Bmatrix} \begin{bmatrix} \underline{\text{GIVING}} \text{ field-name-3} \end{bmatrix}$$

$$\begin{bmatrix} \underline{\text{UNROUNDED}} \end{bmatrix} \begin{bmatrix} ; \text{ ON } \underline{\text{SIZE}} \ \underline{\text{ERROR}} \text{ any imperative statement} \end{bmatrix}$$

Notes:

    1. All notes specified under the ADD verb apply to the MULTIPLY verb.

NOTE

FUNCTION: To allow the programmer to write explanatory material in his program which will be produced on the listing but not compiled.

NOTE . . .

Notes:

1. Following the word NOTE may appear any combination of the allowable character set.

2. If **NOTE** is the first word of a paragraph the entire paragraph must be notes. Proper format rules for paragraph structure must be observed. This paragraph may not be named.

3. If **NOTE** is not the first word of a paragraph, the commentary ends with a period followed by a space.

FUNCTION: To initiate the processing of both input and output files. Performs checking or writing of labels, and other input/ output functions.

OPEN $\left[\underline{\text{INPUT}} \text{ file-name-1} \left[\underline{\text{REVERSED}}\right] \left[, \text{ file-name-2} \ldots\right]\right]$
$\left[\underline{\text{OUTPUT}} \text{ file-name-3} \left[, \text{ file-name-4}\right] \ldots\right]$

Notes:

1. At least one file must be named when the OPEN verb is used.

2. The verb OPEN must be applied to all files and must be executed prior to the first READ or WRITE of this file.

3. A second OPEN of a file cannot be executed prior to the execution of a CLOSE of the file.

4. The OPEN does not obtain or release the first data record. A READ or WRITE respectively must be executed to obtain or release the first data record.

5. When checking or writing the first label, the user's beginning label subroutine will be executed if one is specified by the USE verb.

6. The REVERSED option can only be used on single reel input files.

7. If an input file has been designated as OPTIONAL in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION, the object program will cause an interrogation for the presence or absence of this file to occur. If the reply to the interrogation is negative, i.e. the file is not present, the file will not be OPENED, a print-out indicating the absence of the file will occur, and an "end of file" signal will be sent to the input-output control system of the object program. Thus, when the first READ for this file is encountered, the "end of file" path for this statement will be taken.

FUNCTION: To depart from the normal sequence of procedures in order to execute one statement or a sequence of statements a specified number of times or until a limit is reached, and to provide a means of return to the normal sequence.

PERFORM procedure-name-1 $\left[\underline{\text{THRU}} \text{ procedure-name-2} \right]$

$$\left[\left\{\begin{array}{l}\underline{\text{EXACTLY}} \left\{\begin{array}{l}\text{integer-1} \\ \text{field-name-1}\end{array}\right\} \underline{\text{TIMES}} \\[2ex] \underline{\text{UNTIL}} \text{ condition} \\[2ex] \underline{\text{VARYING}} \text{ subscript-name} \left\{\begin{array}{l}\underline{\text{BY}} \\ \underline{\text{FROM}} \\ \underline{\text{TO}}\end{array}\right\} \left\{\begin{array}{l}\text{integer-2} \\ \text{field-name-2}\end{array}\right\} \\[4ex] \left\{\begin{array}{l}\underline{\text{BY}} \\ \underline{\text{FROM}} \\ \underline{\text{TO}}\end{array}\right\} \left\{\begin{array}{l}\text{integer-3} \\ \text{field-name-3}\end{array}\right\} \left\{\begin{array}{l}\underline{\text{BY}} \\ \underline{\text{FROM}} \\ \underline{\text{TO}}\end{array}\right\} \left\{\begin{array}{l}\text{integer-4} \\ \text{field-name-4}\end{array}\right\}\end{array}\right\}\right]$$

Notes:

1. PERFORM is the means by which loops are written in COBOL. The loop may be executed once or a number of times, as determined by a variety of controls.

2. The first statement of "procedure-name-1" is the point to which sequence control is sent by PERFORM. The return mechanism is automatically inserted as follows:

   a. If "procedure-name-1" is a paragraph-name, and "procedure-name-2" is not specified, -- after the last statement of the "procedure-name-1" paragraph.

   b. If "procedure-name-1" is a section-name, and "procedure-name-2" is not specified, -- after the last statement of the last paragraph of the "procedure-name-1" section.

   c. If "procedure-name-2" is specified and is a paragraph-name, -- after the last statement of the "procedure-name-2" paragraph.

   d. If "procedure-name-2" is specified and is a section-name, -- after the last statement of the last paragraph of the "procedure-name-2" section.

   The last sentence performed in all the above cases must not contain a GO TO verb. There is no necessary relation between "procedure-name-1" and "procedure-name-2" except that a sequence beginning at "procedure-name-1" must proceed to the last statement of "procedure-name-2". In

PERFORM

particular, GO's and PERFORM's may occur between "procedure-name-1" and the end of "procedure-name-2". If there are two or more paths to the end of the loop, "procedure-name-2" must be a paragraph consisting of the verb EXIT, to which all paths must lead.

3. In all cases, after completion of a PERFORM a bypass is automatically created around the return mechanism which had been inserted after the "last statement". Therefore, when no related PERFORM is in progress, sequence control will pass through a "last statement" to the following statement as if no PERFORM had existed.

4. The PERFORM mechanism operates as follows in the different options, with Note 3 above applying to all:

   a. Simple PERFORM (no options) - a return to the statement following the PERFORM is inserted after the "last statement" as defined in Note 2, and sequence control is sent to "procedure-name-1".

   b. EXACTLY--TIMES. The specified number of times must be an integer, assumed positive, and can be zero. The PERFORM mechanism sets up a counter and tests it against the specified number of times before each jump to "procedure-name-1". The return mechanism after the "last statement" sends control to the counting and test. The test sends control to "procedure-name-1" the specified number of times and after the last time sends control to the statement following the PERFORM.

   c. UNTIL condition. This option is very similar to the EXACTLY-- TIMES option, except that there is no counting and evaluation of the condition takes the place of testing against the specified number of times. The condition may be any simple or compound condition, as described in V.3.2; that is, the condition may involve relations and tests. When the condition is satisfied, that is, true, control is transferred to the next statement after the PERFORM. If the condition is true when the PERFORM is entered, no jump to "procedure-name-1" takes place.

   d. VARYING subscript-name. This option is used when the PERFORM is designed to process in a systematic manner a series of quantities distinguished by the subscript being varied. The PERFORM mecha- nism sets the subscript to the specified starting value (the FROM-), prepares to count by the specified increment (the BY-), and to test for the limit (the TO-). The FROM-, BY-, and TO- may be written in any meaningful order. The operation then proceeds as described for EXACTLY--TIMES. The FROM, BY, and TO values must be positive integers. The FROM value cannot be zero, because the first element of a subscripted list is designated to be one. The PERFORM has been completed when adding the increment results in a number greater than the TO value. After completion of a PERFORM VARYING, the subscript-name remains at the last used value.

5. In general, "procedure-name-1" should not be the next statement after the PERFORM. If it is, the result will be that the loop will be traversed one more time than was probably intended, because after the PERFORM has been satisfied control will go to "procedure-name-1" in the normal continuation of sequence.

6. The scope of the PERFORM statements may be nested provided that the nesting is strictly carried out.

The following are examples of correct and incorrect nesting of routines.

CORRECT	INCORRECT
V. PERFORM A THRU B.	V. PERFORM A THRU B.
W. GO TO Z.	W. GO TO Z.
A. · · · · ·	A.
X. PERFORM C THRU D.	X. PERFORM E THRU D.
U. GO TO E.	Y. GO TO B.
E. · · · · · ·	E.
B. · · · · · ·	B.
Z. · · · · · ·	C.
C.	D.
D.	Z.

FUNCTION: To make available the next logical record from an input file and to allow performance of any imperative statement when end of file is detected.

READ file-name RECORD $\left[\underline{\text{INTO}}\text{ data-name}\right]$ $\left[\text{; AT }\underline{\text{END}}\text{ any imperative statement}\right]$

Notes:

1. An OPEN statement for the file must be executed prior to the execution of the first READ for that file.

2. When a file consists of more than one type of logical record, these records automatically share the same memory area. This is equivalent to saying that there exists an implicit redefinition of the area, and only the information which is present in the current record is accessible.

3. The "INTO data-name" option may only be used when the input file contains just one type of record. The "data-name" may be the name of a working storage or output record area. If the format of the "data-name" differs from that of the input record, moving and editing will be performed according to the rules specified for the MOVE verb without the CORRESPONDING option.

     When the "INTO data-name" option is used, the "file-name RECORD" is still available in the input record area.

4. Every READ statement must have an END of file option, either implicitly or explicitly. If the user does not write END, the compiler will examine all other READ statements for the same file. If the word END appears once and only once for a given file, the compiler will append this and its associated "any imperative statement" to each READ for that file which has no explicit END of file option. If more than one, but not all READs for the same file contain the word END, the compiler will indicate an error during compilation.

5. If an OPTIONAL file is not present the "any imperative statement" will be executed on the first READ. The standard end of file procedures will not be performed. (See the OPEN and USE verbs and the FILE-CONTROL Paragraph, ENVIRONMENT DIVISION)

6. After execution of the "any imperative statement" an attempt to perform a READ without the execution of a CLOSE and a subsequent OPEN for this file will constitute an error in the object program. The results of this error are specified by the individual implementor.

7. After recognition of the end of reel, the READ performs the following operations:

> ### READ

a) The standard ending reel label subroutine and the user's ending reel label subroutine (if specified by the USE verb). The order of execution of these two routines is specified by the USE verb.

b) A tape swap.

c) The standard beginning reel label subroutine and the user's beginning reel label subroutine (if specified by the USE verb). The order of execution of these two subroutines is specified by the USE verb.

d) Makes the next unit record available.

STOP

FUNCTION: To halt the computer either permanently or temporarily.

STOP $\left\{ \begin{array}{l} \text{literal} \\ \text{RUN} \end{array} \right\}$

Notes:

1. If the word RUN is used then the ending procedure established by the installation and/or the compiler is instituted.

2. The literal will be communicated to the operator. If the object program environment includes a monitor printer (typewriter, etc.) the literal will be displayed. If the display device is limited to console lights:

    a) if numeric, the literal will be displayed.

    b) if alphabetic the computer will display a number keyed to a list of STOPs produced by the compiler.

    "Pushing the start bar" will cause the execution of the next statement in sequence.

SUBTRACT

FUNCTION: To subtract one or a sum of quantities from a specified quantity and store the result in the last named field or the specified one.

$$\underline{\text{SUBTRACT}} \begin{Bmatrix} \text{literal-1} \\ \text{field-name-1} \end{Bmatrix} \left[, \begin{Bmatrix} \text{literal-2} \\ \text{field-name-2} \end{Bmatrix} \dots \right] \underline{\text{FROM}} \begin{Bmatrix} \text{literal-n} \\ \text{field-name-n} \end{Bmatrix}$$

$$\left[\underline{\text{GIVING}} \text{ field-name-m} \right] \left[\underline{\text{UNROUNDED}} \right]$$

$$\left[; \text{ON } \underline{\text{SIZE}} \ \underline{\text{ERROR}} \text{ any imperative statement} \right]$$

Notes:

1. All notes specified under the ADD verb apply to the SUBTRACT verb.

2. In addition to the above, when dealing with multiple subtrahends, the effect of the subtraction will be as if the subtrahends were first summed and this sum was then subtracted from the minuend.

    (Note: the minuend is $\begin{Bmatrix} \text{literal-n} \\ \text{field-name-n} \end{Bmatrix}$.)

FUNCTION: To specify procedures for any computer I/O error and label handling
which are in addition to the standard procedures supplied by the
input/output system.

Option 1:

USE procedure-name-1 [THRU procedure-name-2] AFTER

STANDARD ERROR PROCEDURE ON $\left\{\begin{array}{l}\text{file-name}\\\text{INPUT}\\\text{OUTPUT}\end{array}\right\}$

Option 2:

USE procedure-name-1 $\left[\text{THRU procedure-name-2}\right]\left\{\begin{array}{l}\text{BEFORE}\\\text{AFTER}\end{array}\right\}$ STANDARD

$\left[\left\{\begin{array}{l}\text{BEGINNING}\\\text{ENDING}\end{array}\right\}\right]\left[\left\{\begin{array}{l}\text{REEL}\\\text{FILE}\end{array}\right\}\right]$ LABEL PROCEDURE ON $\left\{\begin{array}{l}\text{file-name}\\\text{INPUT}\\\text{OUTPUT}\end{array}\right\}$

Notes:

1. The designated statements will be executed by the input/output system
at the appropriate time, that is:

   a) AFTER completing the standard computer I/O error routine, when
   option 1 is used.

   b) BEFORE or AFTER an input label check procedure is accomplished.

   c) BEFORE or AFTER an output label is created, but before writing on
   tape.

2. In Option 2, if BEGINNING or ENDING are not included, the designated
procedures will be executed for both beginning and ending labels. If
REEL and FILE are not included, the designated procedures will be
executed for both REEL and FILE labels.

FUNCTION: To release a logical record for an output file and to allow for vertical positioning if the output medium is an on-line printer.

WRITE record-name $\left[\underline{\text{FROM}}\text{ data-name}\right]$

$$\left[\left\{\begin{array}{l}\underline{\text{AFTER}}\\\underline{\text{BEFORE}}\end{array}\right\}\underline{\text{ADVANCING}}\left\{\begin{array}{l}\text{field-name}\\\text{integer}\end{array}\right\}\text{ LINES}\right]$$

Notes:

1. After the WRITE is given, "record-name" is no longer available.

2. The "data-name" may be the name of working storage or an input record area. If the format of the "data-name" differs from that of the "record-name", moving and editing will take place according to the rules specified for the MOVE verb without the CORRESPONDING option.

3. An OPEN statement must be executed prior to giving the first WRITE for that file.

4. The ADVANCING option allows vertical positioning of each record on the printed page. The "field-name" must have an integral value.

5. After recognition of the end of reel, the WRITE performs the following operations:

    a) The standard ending reel label subroutine and the user's ending reel label subroutine (if specified by the USE verb). The order of execution of these two subroutines is specified by the USE verb.

    b) A tape swap.

    c) The standard beginning reel label subroutine and the user's beginning reel label subroutine (if specified by the USE verb). The order of execution of these two subroutines is specified by the USE verb.

7. LIST OF KEY AND OPTIONAL WORDS IN THE PROCEDURE DIVISION

7.1 KEY WORDS

7.1.1 <u>Key Words For Verbs</u>

The key words appearing in the verb formats are:

ACCEPT	INPUT
ADD	INTO
ADVANCING	LABEL
AFTER	LEADING
ALL	LOCK
ALTER	MOVE
AS	MULTIPLY
BEFORE	NO
BEGINNING	NOTE
BY	OPEN
CLOSE	OUTPUT
COBOL	PERFORM
COMPUTE	PROCEED
CORRESPONDING	READ
DEFINE	REEL
DEPENDING	REPLACING
DISPLAY	REVERSED
DIVIDE	REWIND
END	RUN
ENDING	SIZE
ENTER	STOP
EQUALS	SUBTRACT
ERROR	TALLYING
EXACTLY	THROUGH
EXAMINE	THRU
EXIT	TIMES
FILE	TO
FILLING	UNROUNDED
FIRST	UNTIL
FORMAT	UPON
FROM	USE
GIVING	VARYING
GO	VERB
HERE	WRITE
INCLUDE	

THROUGH THRU } INTERCHANGEABLE

7.1.2 Additional Key Words

AND
ALSO
DIVIDED
DIVISION
EQUAL
EXCEEDS
EXPONENTIATED
FOR
GREATER
HIGH-VALUE
HIGH-VALUES
IF
IN
LESS
LOW-VALUE
LOW-VALUES
MINUS
MULTIPLIED
NEGATIVE
NOT
NUMERIC
OF
OR
OTHERWISE
PLUS
POSITIVE
SECTION
SPACE
SPACES
TALLY
THEN
UNEQUAL
ZERO
ZEROES
ZEROS

7.2 OPTIONAL WORDS

The optional words are:

AT
IS
LINES
ON
PROCEDURE
RECORD
STANDARD
THAN
WITH

VI. DATA DIVISION

1. GENERAL DESCRIPTION

1.1 OVERALL APPROACH

Data to be processed falls into three categories - that which is contained in files and enters or leaves the internal memory of the computer from specified areas, that which is developed internally and placed into intermediate or working storage, and constants which are defined by the user. (Figurative constants and literals used in procedure statements are not listed in the DATA DIVISION.) Tables may fall into any of the above categories.

The approach taken in defining file information is to distinguish between the physical aspects of the file (i. e., the File Description) and the conceptual characteristics of the data contained therein (i. e., the Record Description). By physical aspects is meant the mode in which the file is recorded, the grouping of logical records within the physical limitations of the file-media, the means by which the file can be identified, etc. By conceptual characteristics is meant the explicit definition of each logical entity within the file itself.

For purposes of processing, the contents of a file are divided into logical records. By definition, a logical record is any consecutive set of information. In an Inventory Transaction File, for example, a logical record could be defined as a single transaction, or as all consecutive transactions which pertain to the same stock item. It is important to note that several logical records may occupy a block (i. e., physical record), or a logical record may extend across physical records.

The concept of a logical record is not restricted to file data, but is carried over into the definition of working storages and constants. Thus, working storages and constants may be grouped into logical entities and defined by a Record Description.

File Description and Record Descriptions may be stored on a COBOL library tape.

1.2 ORGANIZATION

The DATA DIVISION, which constitutes the second division of the problem definition, is subdivided according to types of data. That is, it consists of a FILE Section, a WORKING-STORAGE Section and a CONSTANT Section.

The DATA DIVISION contains two elements: descriptions of files and descriptions of records. Both of these elements appear within the FILE Section. The FILE Section contains File Descriptions and Record Descriptions for both label records and data records. These two kinds of records are defined in the same manner; however, because the input-output system of the object program must perform special operations on label records, fixed names have been assigned to those label fields on which specified operations must be performed.

The WORKING-STORAGE and CONSTANTS sections consist solely of Record Descriptions and unrelated Record Description entries.

1.3 STRUCTURE

The DATA DIVISION begins with the header:

DATA DIVISION [PREPARED FOR computer-name] .

The optional clause is only used when the data descriptions were written for other than the object computer. Each of the three sections begins with the appropriate section name followed by the word SECTION and a period - i.e., FILE, WORKING-STORAGE, or CONSTANT. When a section is not required, its name need not appear.

The sections themselves consist of entries. This is a departure from the paragraph-sentence-statement structure of the PROCEDURE and EN-VIRONMENT DIVISIONS.

An entry consists of a level indicator, a data-name and a series of independent clauses which may be separated by semicolons. The clauses may be written in any sequence, except when otherwise specified in the entry formats. The entry itself is terminated by a period. A File Description consists of a single entry, whereas, a Record Description consists of one or more entries.

A technical discussion of the reference format for the DATA DIVISION is given in Chapter VIII. 3 of this report.

2. FILE DESCRIPTION ENTRY

2.1 GENERAL DESCRIPTION

A File Description entry contains information pertaining to the physical aspects of a file. In general it may include the following: the manner in which the data is recorded on the file, the volume of data in the file, the size of the logical and physical records, the names and values of the label records contained in the file, the names of the data records which comprise the file, and finally, the keys on which the data records have been sequenced.

The listing of data and label record names in a File Description entry serves as a cross reference between the file and the records in the file. If the Record Description for these records is not found within the DATA DIVISION of the problem description, they are automatically called from the COBOL library.

2.2 ENTRY FORMATS

2.2.1 General Notes

A File Description entry consists of a level indicator, a file name, and a series of independent clauses which define the physical and logical characteristics of the file. The mnemonic level indicator FD is used to identify the start of a File Description entry, and distinguishes this entry from those associated with a Record Description.

2.2.2 Specific Formats

The individual clause formats are arranged in this report in alphabetic order where as the clauses in the "Complete Entry" are shown in the recommended order.

> FILE DESCRIPTION
> Complete Entry

FUNCTION: To furnish information concerning the physical structure, identification and record descriptions pertaining to a given file.

Option 1:

FD file-name COPY library-name.

Option 2:

FD file-name [; RECORDING MODE IS mode]

[; FILE CONTAINS ABOUT integer-1 RECORDS]

$$\left[; \text{BLOCK CONTAINS } [\text{integer-2 } \underline{TO}] \text{ integer-3 } \left\{\begin{array}{l}\underline{RECORDS}\\ \underline{CHARACTERS}\end{array}\right\}\right]$$

[; RECORD CONTAINS [integer-4 TO] integer-5 CHARACTERS]

$$; \underline{LABEL} \underline{RECORDS} \left\{\begin{array}{l}ARE\\ IS\end{array}\right\} \left\{\begin{array}{l}\underline{STANDARD}\\ \underline{OMITTED}\\ \text{data-name-1}\\ \text{library-name-1 IN } \underline{LIBRARY}\end{array}\right\}$$

$$\left[, \left\{\begin{array}{l}\text{data-name-2}\\ \text{library-name-2 IN } \underline{LIBRARY}\end{array}\right\} \cdots\right]$$

$$\left[; \underline{VALUE} \text{ OF field-name-1 IS } \left\{\begin{array}{l}\text{literal}\\ \text{data-name-3 } [\underline{HASHED}]\end{array}\right\} [, \text{ field-name-2 } \ldots .]\right]$$

$$; \underline{DATA} \underline{RECORDS} \left\{\begin{array}{l}ARE\\ IS\end{array}\right\} \left\{\begin{array}{l}\text{data-name-4}\\ \text{library-name-3 IN } \underline{LIBRARY}\end{array}\right\}$$

$$\left[, \left\{\begin{array}{l}\text{data-name-5}\\ \text{library-name-4 IN } \underline{LIBRARY}\end{array}\right\} \cdots\right]$$

$$\left[; \underline{SEQUENCED} \text{ ON field-name-3 } [, \text{ field-name-4 } \ldots .]\right].$$

Notes:

1. Option 1 is used when the COBOL library contains the entire File Description entry.

2. The level indicator FD identifies the beginning of the file description entry. As such, it precedes the file-name and appears at the left margin of the reference listing. (See Chapter VIII. 3, REFERENCE FORMAT, DATA DIVISION.)

```
FILE DESCRIPTION
  Complete Entry
```

3. The clauses which follow the name of the file are optional in many cases. For further details, see the individual explanations for each clause.

4. The File Description entry is terminated by a period.

5. All semicolons are optional in the File Description.

6. In the Reference Format, those entries which require more than a single line are continued on subsequent lines starting under the first letter in the file name. (See Chapter VIII. 3, REFERENCE FORMAT, DATA DIVISION.)

FUNCTION: To specify the size of the physical record (i.e., block).

$$\left[\; ; \; \underline{\text{BLOCK}} \; \text{CONTAINS} \; \left[\; \text{integer-2} \; \underline{\text{TO}} \; \right] \; \text{integer-3} \; \left\{ \begin{array}{l} \underline{\text{RECORDS}} \\ \text{CHARACTERS} \end{array} \right\} \; \right]$$

Notes:

1. This clause is required except when one of the following exists:

    a. A physical record contains one and only one complete logical record.

    b. The object computer has one and only one physical record size.

    c. The object computer has more than one physical record size but the implementor has designated one size as Standard. In this case, the absence of this clause denotes the Standard physical record size.

2. When this clause is used, the size may be stated in terms of RECORDS, unless one of the following situations exists; in which case the CHARACTERS' option should be used:

    a. Logical records extend across physical records.

    b. Physical record contains padding; i.e., area not contained in a logical record.

    c. Logical records are grouped in such a manner that an inaccurate physical record size would be implied.

3. When the CHARACTERS' option is used, the physical record size is specified in terms of the number of BASIC characters which could be contained within the physical record regardless of the way in which the information is recorded. See VI 3.1.6 for definition of BASIC.

4. If only "integer-3" is shown then it represents the exact size of the physical record. If "integer-2" and "integer-3" are both shown, then they refer to the minimum and maximum size of the physical record respectively.

5. The word CHARACTERS within the BLOCK clause is an optional word as it is everywhere else. Whenever the key word RECORD is not specifically written in the BLOCK clause, "integer-2" and "integer-3" represent CHARACTERS.

FUNCTION: To obtain a file description entry from the COBOL library.

<u>COPY</u> library-name

Notes:

1. COPY is used when the COBOL library contains the entire File Description entry.

2. During compilation, the COPY clause is replaced by the sequence of clauses within the "library-name" entry. Thus, the level indicator and file-name which precede the COPY clause will replace the level indicator and file-name appearing within the "library-name" entry.

```
┌─────────────┐
│ DATA        │
│ RECORDS     │
└─────────────┘
```

FUNCTION: To crossreference the description of data records associated with the file.

$$; \underline{\text{DATA}} \ \underline{\text{RECORDS}} \ \begin{Bmatrix} \text{ARE} \\ \text{IS} \end{Bmatrix} \begin{Bmatrix} \text{data-name-4} \\ \text{library-name-3 IN } \underline{\text{LIBRARY}} \end{Bmatrix}$$
$$\left[\ , \begin{Bmatrix} \text{data-name-5} \\ \text{library-name-4 IN } \underline{\text{LIBRARY}} \end{Bmatrix} \cdots \right]$$

Notes:

1. This clause is required in every File Description entry.

2. The presence of more than one data name indicates that the file contains more than one type of data record. These records may be of differing sizes, different formats, etc. The order in which they are listed is not significant.

3. Conceptually, all data records within a file share the same area. This is in no way altered by the presence of more than one type of data record within the file.

4. If the IN LIBRARY option is used, the Record Description is in the COBOL library and need not be written in the program.

5. If logical records of different sizes are grouped into one physical record, then, the end of the logical record must be explicitly defined in the Record Description. See VI 3.2.2. SIZE.

FILE Size

FUNCTION: To indicate the approximate number of logical records in a file.

$$\left[\; ; \; \underline{\text{FILE}} \; \text{CONTAINS ABOUT integer-1} \; \underline{\text{RECORD}}\text{S} \; \right]$$

Note:

1. This optional clause has been included for potential use in the language.

```
┌──────────┐
│ LABEL    │
│ RECORDS  │
└──────────┘
```

FUNCTION: To cross reference the descriptions of the label records associated with the file.

$$; \underline{\text{LABEL}} \ \underline{\text{RECORDS}} \ \begin{Bmatrix} \text{ARE} \\ \text{IS} \end{Bmatrix} \begin{Bmatrix} \underline{\text{STANDARD}} \\ \underline{\text{OMITTED}} \\ \text{data--name--1} \\ \text{library--name--1 IN} \ \underline{\text{LIBRARY}} \end{Bmatrix}$$

$$\left[\ , \ \begin{Bmatrix} \text{data--name--2} \\ \text{library--name--2 IN} \ \underline{\text{LIBRARY}} \end{Bmatrix} \ \cdots \ \right]$$

Notes:

1. This clause is required in every File Description entry. When a file contains no beginning or ending, tape or file label, the word OMITTED must be used.

2. STANDARD implies a set of label description formats which are defined by the implementor and, therefore, need not be written in the program.

3. The following four types of label records may appear on the tapes associated with a file. Since the type of label is significant, the fixed record names shown in capital letters have been assigned:

    a) A BEGINNING-TAPE-LABEL which appears at the beginning of each tape and which precedes all other information, and contains information about the tape.

    b) A BEGINNING-FILE-LABEL which appears only once and which precedes the first data record in the file but follows the beginning tape label if one is present. This label contains information about the file.

    c) An ENDING-TAPE-LABEL which immediately follows the last valid data or label record on the tape. The end of the tape label must appear before the physical end of the tape is encountered. When both end of file and end of tape labels are being employed, the end of tape label will follow the end of file label. On a MULTIPLE-FILE-TAPE, when all four types of labels are being employed, the end of tape label follows the last end of file label; all other end of file labels are followed by the next beginning file label. This label contains information about the tape.

    d) An ENDING-FILE-LABEL which appears only once and which immediately follows the last data record in the file, and contains information about the file.

```
┌─────────────┐
│ LABEL       │
│ RECORDS     │
└─────────────┘
```

4. Many files employ only two types of labels, namely, beginning and ending tape labels. Instead of having separate file and tape labels, the beginning tape labels contain all pertinent information normally found in the beginning file label, and the ending reel label contains a sentinel field for which special end of tape and end of file conditions are defined.

5. All other types of labels (i. e., additional labels and labels which are radically different from those defined above), must be defined as data rather than label records. All such records must be listed in the "DATA RECORDs" clause rather than in the "LABEL RECORDs" clause. These other labels and the method of handling them will be specified by individual implementors.

6. When the IN LIBRARY option is used, the Record Description is in the COBOL library and need not be written in this program.

RECORD Size

FUNCTION: To specify the size of data records.

$$\left[\underline{\text{RECORD}} \text{ CONTAINS } \left[\text{integer-4 } \underline{\text{TO}} \right] \text{ integer-5 CHARACTERS} \right]$$

Notes:

1. The size of each data record is completely defined within the Record Description entries, therefore this clause is never required. When present, however, the following notes must apply.

2. "Integer-5" may not be used by itself unless all the data records in the file have the same size. In this case then "integer-5" represents the exact number of characters in the data record. If "integer-4" and "integer-5" are both shown, then they refer to the minimum number of characters in the smallest size data record and the maximum number of characters in the largest size data record, respectively.

3. Characters are specified in terms of the number of BASIC characters, which could be contained within the data record, regardless of the type of character actually employed. See VI 3.1.6 for definition of BASIC.

RECORDING
Mode

FUNCTION: To specify the organization or type of data as it exists on the external media.

$$\left[\underline{\text{RECORDING}} \text{ MODE IS mode} \right]$$

Notes:

1. The RECORDING MODE clause is necessary for those computers whose organization of data, as it exists on the external media, may vary.

2. When a computer is capable of having only one mode, this clause is not needed.

3. Each implementor will assign specific names to the alternative modes of data representation which he is prepared to handle. When a standard RECORDING MODE exists, the implementor may choose to assign names to only the non-standard modes. In this case, the absence of the RECORDING MODE clause denotes the standard mode.

SEQUENCED

FUNCTION: To indicate the keys on which the data records are sequenced.

$$\left[\; ; \; \underline{\text{SEQUENCED}} \text{ ON field-name-3} \left[\; , \text{ field-name-4} \; . \; . \; . \right] \right]$$

Notes:

1. The SEQUENCED clause is optional.

2. "field-name-3" represents the major key, "field-name-4" represents the next highest key, etc.

3. This clause does not imply an automatic sequence check at object time.

VALUE

FUNCTION: To particularize the description of the fields in the label records associated with the file.

$$\left[\text{; } \underline{\text{VALUE}} \text{ OF field-name-1 IS } \begin{Bmatrix} \text{literal} \\ \text{data-name-3 } \left[\underline{\text{HASHED}} \right] \end{Bmatrix} \left[\text{, field-name-2 IS } \dots \right] \right]$$

Notes:

1. When the HASHED option is not used and when the file in question is being processed as an input, the appropriate label check routine (i.e., the beginning or ending file or tape label check routine) will check for equality between "field-name-1" and the "literal" or "data-name-3" whichever is specified. When the file is being processed as an output file, the value of the "literal" or "data-name-3" will, at the appropriate time, be positioned and written as "field-name-1". The VALUE clause must be used without the HASHED option when specifying the required values of such fields as File-name, File-number, etc.

2. When the HASHED option is specified and when the file is processed as an input, a hash total of "data-name-3" is kept and at the appropriate time, is checked against the "field-name-1". When the file is being processed as an output, a hash total of "data-name-3" is kept and at the appropriate time, it is positioned and written as "field-name-1". When the HASHED option is employed, "field-name-1" must appear in an ending rather than in a beginning label.

3. RECORD DESCRIPTION

3.1 GENERAL DESCRIPTION

3.1.1 Elements of Record Description

A Record Description consists of a set of entries. Each entry defines the characteristics of a particular unit of data. With minor exceptions, each entry is capable of completely defining a unit of data. Because the COBOL Record Descriptions involve a hierarchal structure (i. e., an entry giving only the general characteristics may be followed by a set of subordinate entries which together redescribe the unit in more specific terms), the contents of an entry may vary considerably, depending upon whether or not it is followed by subordinate entries. For further explanation of subordinate entries, see Chapter VI, 3.1.3 "Concept of Level."

In defining the lowest level or subdivision of data, (i. e., a field or sub-field), the following information may be required:

a) A level number which shows the relationship between this and other units of data.

b) A data-name.

c) The SIZE in terms of numbers of characters.

d) The BASE in which the characters are represented.

e) The number of consecutive occurrences (OCCURS) of the same unit of data.

f) The RANGE of values which the data may assume.

g) The CLASS or type of data - i. e., alphabetic, numeric or alphanumeric.

h) The location and type of SIGN.

i) Location of actual or assumed radix point.

j) Location of editing symbols such as dollar signs and commas.

k) Justification and Synchronization of data.

l) Special editing requirements such as zero suppression and check protection.

m) Initial VALUE of a working storage or the fixed VALUE of a constant.

An entry which defines a unit of data may not be contradicted by a subordinate entry. Thus, once the BASE is defined, it applies to all subordinate entries and need not be re-specified. Similarly, when CLASS is defined as

alphanumeric, subordinate entries may particularize the class by specifying alphabetic or numeric; but when defined as alphabetic or as numeric, subordinate entries may not change the CLASS.

3.1.2 Concept of Computer Independent Record Descriptions

For a record description to be computer independent, it must apply to all computers which can accept the same input media. In addition, it should apply to all computers for which the basic data can be converted (i. e., for computers which can not accept the same data, but for which the data can be re-recorded in an acceptable form).

These goals are partially achieved through the single or combined use of two types of record descriptions, namely, a definition of the data as it exists on the external media and/or a definition of the data as it exists in the internal memory of the computer. At the present time, the rules for specifying the former are not completely defined. For this reason, the remainder of the report will pertain to the specification of the data representation within the computer.

Commonality is further advanced by the definition of numeric quantities in decimal terms regardless of the computer's internal representation.

Compatibility among computers is not guaranteed, however it may be achieved in some cases with careful planning in the layout of the data.

3.1.3 Concept of Levels

A hierarchal organization of data is inherent within the concept of a logical record. A record may be subdivided into smaller groups of data, which in turn may be subdivided into still smaller groups of data, etc.

For example, consider a file of job tickets sorted (from major to minor) according to division, department, employee number, and day of the week. If the logical record has been defined as all consecutive data pertaining to a single employee, the following levels could be defined:

a) A weekly job record which consists of--

b) Daily job ticket groupings (i. e., subrecords) which consist of--

c) Job tickets (i. e., sub-records) which consist of--

d) The individual fields within the job ticket.

Within a COBOL Record Description, the programmer organizes and defines data according to its relative level. That is, separate entries are written for each level and for each item of data within each level. The definition of a particular item of data consists of the entry written for that level plus all following entries which are of a lower level. The level, itself, is shown by a level number which is relative to the largest element of data within the Record Description (i. e., to the record itself). Level numbers start at 1 (for records) and may go as high as 49. Thus, it is possible to define 49 levels of data but it is not expected that any problems will use this many.

Two types of data exist for which there is no true concept of level, namely non-contiguous constants or working storage fields and the names of conditions associated with a previously defined field.

Non-contiguous constants which bear no relationship to one another and which may not be further subdivided have been assigned the special level number 77.

Entries which specify condition names, to be associated with a particular value of a field, and do not themselves introduce data, have been assigned the special level number 88. For further details concerning the definition of condition names, see 3.2.3 in this Chapter, "Specific Entry For Condition Names".

3.1.4 Concept of Mapping

The essence of defining data as it appears in the internal memory lies in the mapping of consecutive character positions. By mapping is meant – that each accessible memory position, within a logical record, must be accounted for in sequence. Thus, in addition to pertinent characters, all unused or non-significant characters must be defined.

To aid in achieving compatibility between Record Description for different computers, zeroes or spaces which precede or follow a field which is isolated in a separate computer word or words, may be implied rather than explicitly defined. Such zeroes or spaces are implied by stating that the field in question is "synchronized". In this case, the remainder of the word is assumed to be zero or space filled represented in the same base as that of the field. Note, implied zeroes and spaces are not counted in the specification of such fields.

In mapping memory, a clear distinction must be drawn between "a computer sign position" which is capable of receiving any type of character and one which is not. If a "computer sign position" is capable of receiving any type of character, it is considered a "mappable" position, and is treated as any other character position. If the "computer sign position" may not receive any type of character, the sign position is considered "unmappable".

3.1.5 Basic Concepts of Signs

There are several aspects involved in the handling of signs, and they are all interrelated. Thus, a definition and understanding of any one of these is not sufficient to understand the whole picture. Although only one of the entries in the Record Description pertains primarily to this problem, it is of sufficient general importance to warrant a discussion of it.

The following concepts are involved in the specification of signs: mappable and unmappable signs, "computer sign position", and "standard sign position".

A sign is considered mappable if it appears in a memory position capable of containing any type of character. In this case, it may appear in any position within the data record and is included when specifying the size. A sign is considered unmappable if it appears in a position which is incapable of containing every type of character, and is not included in the specification of the size.

Mappable signs may appear anywhere in the data in a computer which has an un-mappable sign. Unmappable signs, however, may only be used on computers with unmappable signs.

The second factor involved in sign specification is the "computer sign position". In fixed word length computers, this means the position in which the sign associated with each computer word may appear. In variable word length computers, it is the normal sign position associated with each field.

The "standard sign position" in a fixed word length computer is defined to be the "computer sign position" associated with the word containing the most significant portion of the field if the field extends across more than one computer word. In a variable word length computer, the "standard sign position" is the normal position associated with the entire field.

Non-standard unmappable signs cannot be defined.

The sign can be defined as part of the current field only if all the following conditions are satisfied: it is represented in the same BASE as the field, no other fields separate it from the current field, and the sign does not apply to more than one field.

3.1.6 Concept of Character Base

Within the Record Description, data size is expressed in terms of characters. Because many computers use more than one type of character (for example, binary coded decimal and pure binary), it is essential that the type of character (i.e., base) be known.

At the present time, two different bases are permitted for each computer: BASIC and OTHER. The actual character representation to be associated with each of these will depend upon the implementor. However, the following criteria governs the implementor's choice.

A BASIC character is one which is capable of representing any alphabetic, numeric or special character existing within the basic character set of the computer. In most instances, BASIC will apply to binary coded decimal characters which can be keypunched, read, written and/or printed.

An OTHER character is one whose representation differs from that of a BASIC character, and which is capable of representing any numeric digit. Normally, OTHER applies to an alternative character representation whose primary function is to facilitate efficient computation.

Note: for those computers which permit only one type of character representation, the base need not be defined.

3.2 ENTRY FORMATS

3.2.1 General Notes

A Record Description consists of a set of entries. Each record description entry, itself, consists of a level number, a data name and a series of independent clauses.

3.2.2 Specific Formats

The individual clause formats are arranged in this report in alphabetic order whereas the clauses in the "Complete Entry Skeleton" are shown in the recommended order.

FUNCTION: To specify the characteristics of a particular unit of data.

Option 1:

      level-number data-name $\left[\text{; } \underline{\text{REDEFINES}} \ldots \right]$; $\underline{\text{COPY}} \ldots$.

Option 2:

      level-number $\left\{ \begin{matrix} \underline{\text{FILLER}} \\ \text{data-name} \end{matrix} \right\}$ $\left[\text{; } \underline{\text{REDEFINES}} \ldots \right]$ $\left[\text{; } \underline{\text{SIZE}} \ldots \right]$

      $\left[\text{; } \underline{\text{BASE}} \ldots \right]$ $\left[\text{; } \underline{\text{OCCURS}} \ldots \right]$ $\left[\text{; } \underline{\text{SIGN}} \ldots \right]$

      $\left[\text{; } \underline{\text{SYNCHRONIZED}} \ldots \right]$ $\left[\text{; } \underline{\text{POINT}} \ldots \right]$

      $\left[\text{; } \underline{\text{CLASS}} \ldots \right]$ $\left[\text{; } \underline{\text{PICTURE}} \ldots \right]$ $\left[\text{; } \underline{\text{JUSTIFIED}} \ldots \right]$

      $\left[\text{; } \underline{\text{ZERO}} \ \underline{\text{SUPPRESS}} \ldots \right]$

      $\left[\text{; } \underline{\text{RANGE}} \ldots \right]$ $\left[\text{; } \underline{\text{VALUE}} \ldots \right]$.

Notes:

1. For detailed explanation of reference format, see Chapter VIII. 3.

2. Those clauses which begin with SIGN, SYNCHRONIZED, POINT, PICTURE, JUSTIFIED, ZERO SUPPRESS, BLANK, RANGE, and VALUE, may not be specified except at the field or sub-field level. Because there is no specific identification for a field in a Record Description entry, "field level" may be defined as the highest level at which a single (alphabetic or numeric) quantity having a homogeneous BASE can be specified.

3. All semicolons are optional in the Record Description entry.

BASE

FUNCTION: To specify the type of character representation.

$$\left[\quad \underline{\text{BASE}} \text{ IS } \left\{ \begin{array}{l} \underline{\text{BASIC}} \\ \underline{\text{OTHER}} \end{array} \right\} \text{ TYPE} \quad \right]$$

Notes:

1. BASE is never required when the data is to be processed on computers having only one set of character representation.

2. BASE is required at the lowest level of data being described. When BASE is used, it applies to all following entries of the same or lower levels until the word BASE is written again. The BASE given at any level must not contradict the BASE at a higher level which contains the data.

3. If the data consists of characters having different bases, it is called non-homogeneous. The BASE option may not be used to describe non-homogeneous data.

4. The actual character representation associated with the base codes BASIC and OTHER will be specified by the implementor, according to the following criteria:

   a) A BASIC character is one which is capable of representing any alphabetic, numeric or special character existing within the basic character set.

   b) An OTHER character is one whose representation differs from a BASIC character, and which is capable of representing – at least each numeric digit. Normally, OTHER applies to an alternative representation whose primary function is to facilitate efficient computation.

FUNCTION: To indicate the type of data being described.

$$\left[\quad \underline{\text{CLASS}} \text{ IS} \quad \left\{ \begin{array}{l} \underline{\text{ALPHABETIC}} \\ \underline{\text{NUMERIC}} \\ \underline{\text{ALPHANUMERIC}} \\ \underline{\text{AN}} \end{array} \right\} \quad \right]$$

Notes:

1. AN is an acceptable abbreviation for ALPHANUMERIC.

2. The CLASS option is required at the lowest level if a field PICTURE has not been given. When CLASS is used, it applies to all following entries of the same or lower levels until a different class is specified, either by the CLASS option or by the field PICTURE. An error will occur if the information in a CLASS option or PICTURE contradicts the CLASS of a higher level. Thus, when CLASS is defined as alphanumeric, lower level entries may particularize the class by specifying alphabetic or numeric; however, when defined as alphabetic or as numeric, lower level entries may not change the CLASS.

FUNCTION: To duplicate within this record all or part of another Record
Description.

$$\Big[\ \underline{COPY} \ \text{data-name} \ \Big]$$

Notes:

1. The information to be duplicated begins following the data-name in the
"data-name" entry. That is, it excludes the level number and the data-
name itself. The remainder of the "data-name" entry and all lower level
entries are then duplicated with corresponding level number adjustments
being applied to the level entries being duplicated to reflect the data-
name level heirarchy. That is, all levels within the "data-name" being
copied are considered relative level numbers rather than absolute level
numbers. Duplication ends at the occurrence of a level number which is
equal to or less than the original level number associated with the "data-
name" being copied.

2. The duplication itself begins at the point in this entry where the COPY
clause appears. Thus, the level number, data-name, and all preceding
clauses are not altered by the duplication.

> **Data-Name**
> **FILLER**

FUNCTION: To specify the name of the data being described, or to specify an unused portion of the logical record.

$$\left\{ \begin{array}{c} \text{data-name} \\ \underline{\text{FILLER}} \end{array} \right\}$$

Notes:

1. The "data-name" or FILLER is required as the second element in each Record Description entry.

2. Qualification of the "data-name" is automatically provided through higher level data-names and file-names. Thus "data-name" need not be unique within or between Record Descriptions provided a higher level data-name or a file-name can be used for qualification.

3. A FILLER entry describes an area in the record whose contents are immaterial. However, the contents of the FILLER area must be left unaltered by the object program. The area defined by FILLER is not addressable but its size and BASE must be specified.

4. The "data-name" may be a fixed name assigned to a label record or a particular field within a label record. A list of these fixed names and their significance is given VI. 4.1 "File Section".

FUNCTION: To permit suppression of non-significant zeroes and commas, to permit floating dollar signs or check protection, and to permit the blanking of a field when its value is zero.

$$\left[\begin{Bmatrix} \underline{ZERO}\ \underline{SUPPRESS} \\ \underline{CHECK}\ PROTECT \\ \underline{FLOAT}\ DOLLAR\ \underline{SIGN} \end{Bmatrix} \left[\underline{LEAVING}\ integer\ PLACES \right] \right] \left[\underline{BLANK}\ WHEN\ \underline{ZERO} \right]$$

Notes:

1. The rules for editing, as shown in the MOVE verb, specify that fields are moved in conformity with the Record Description format of the receiving field. This format is composed of the Editing options and the field PICTURE.

2. The three options, ZERO SUPPRESS, CHECK PROTECT and FLOAT DOLLAR SIGN, all permit suppression of leading zeroes and commas. If the LEAVING option is not employed, suppression will stop as soon as either a non-zero digit or the decimal point (actual or assumed) is encountered. Specifically,

   a) When ZERO SUPPRESS is specified, leading zeroes and commas will be replaced by spaces.

   b) When CHECK PROTECT is specified, leading zeroes and commas will be replaced by asterisks.

   c) When FLOAT DOLLAR SIGN is specified, the rightmost character suppressed will be replaced by a dollar sign, and all other characters which are suppressed will be replaced by spaces.

3. The LEAVING option may be employed to stop suppression before the decimal point (actual or assumed) is encountered. When used, suppression stops at the character just prior to the "integer" position, unless stopped sooner by the rules specified in Note 2. The "integer" position is a count of the number of characters, starting immediately at the left of the actual or assumed decimal point.

4. When the BLANK WHEN ZERO clause is used, the field will contain nothing but spaces if the value of the field is zero. Thus, all other editing requirements, such as zero suppress, check protect, etc., will be overridden.

5. Because spaces, dollar signs, and asterisks are considered alphabetic characters, the above options may not be specified unless the field is composed of BASIC characters.

VI-27

JUSTIFIED

FUNCTION: To specify non-standard positioning of data within a field when less than the maximum number of characters may be present.

$$\left[\underline{\text{JUSTIFIED}} \left\{ \begin{array}{l} \underline{\text{LEFT}} \\ \underline{\text{RIGHT}} \end{array} \right\} \right]$$

Notes:

1. This option is only required when the standard rules of positioning are not desired. The standard rules of positioning within a field are:

   a. Numeric data is always right justified with zero fill.

   b. Alphabetic and alphanumeric data is left justified with space fill.

page VI-28

<div style="border:1px solid black; display:inline-block; padding:4px;">Level Number</div>

FUNCTION: To show the hierarchy of data within a logical record. To identify entries for condition-names, non-contiguous constant and working storage fields.

<div style="text-align:center;">level-number</div>

Notes:

1. The "level-number" is required as the first element in each Record Description entry.

2. The "level-number" may have values of 1-49 and 77 and 88.

3. The level number 1 signals the first entry in each Record Description. This corresponds to the logical record on which the READ and WRITE verbs operate.

4. Special level numbers have been assigned to certain entries where there is no real concept of level:

    (a) Level number "77" is assigned to identify non-contiguous constants and working storage fields.

    (b) Level number "88" is assigned to entries which define condition-names associated with a conditional variable.

5. For general discussion of the level-number, see Chapter VI. 3.1.3, "Concept of Levels"

OCCURS

FUNCTION: To eliminate the need for separate entries for repeated data and to supply information required in the application of subscripts.

$$\left[\underline{\text{OCCURS}} \; \left[\text{integer-1} \; \underline{\text{TO}} \; \right] \; \text{integer-2} \; \text{TIMES} \right.$$

$$\left. \left[\underline{\text{DEPENDING}} \; \text{ON} \; \left\{ \begin{array}{l} \text{field-name} \\ \text{condition-name} \end{array} \right\} \right] \right]$$

Notes:

1. The OCCURS clause is used in defining tables and other homogeneous sets of data. When OCCURS is used, data defined in the current and all lower levels must be subscripted when referenced, unless "integer-2" equals one.

2. This clause is required either when the data might not exist, or when it might occur more than once. If the clause is not used, the number of occurrences is assumed to be one.

3. If only "integer-2" is shown it represents the exact number of occurrences. It is illegal to have "integer-2" equal to zero.

4. If both "integer-1" and "integer-2" are shown, they refer to the minimum and maximum occurrences respectively. "Integer-2" must always be greater than "integer-1". When "integer-1" is one, the data will be present, but need appear only once. If "integer-1" is zero, the data might not be present.

5. The DEPENDING option is only required when the end of the occurrences cannot otherwise be determined.

6. The use of "field-name" means that the count of the number of occurrences of the data is contained within the data unit called "field-name". This value must appear and be an integer. "Field-name" must appear within the record to which the current Record Description entry pertains. Furthermore, "field-name" must precede "integer-1" occurrences so that the count can always be found.

7. The use of "condition-name" within the DEPENDING option implies that when the condition is satisfied, the occurrences terminate. The "condition-name" must be associated with a field which appears within the unit which is being repeated. For discussion of condition-names see III. 2. 2. 1 (b).

PICTURE

FUNCTION: To show a detailed picture of the field structure and permit editing representation.

[PICTURE IS any combination of allowable characters described below]

Notes:

1. The PICTURE is required for fields containing editing symbols such as the dollar sign, comma, and/or actual decimal point and mappable signs which have not been specified in the SIGN clause.

2. If the PICTURE is shown at any level, then it must be shown at each lower level.

3. The allowable characters and their respective definitions are as follows:

   A represents an alphabetic character
   X represents an alphanumeric character
   9 represents a numeric character

   V represents the assumed decimal point and is not a mappable character
   P represents a scaling position that is used in locating an assumed decimal point outside the field and it is not a mappable character
   - indicates the negative sign only
   S indicates the presence of a positive or negative sign
   I or + 9 represents a numeric character with a sign always overpunched
   R or -9 represents a numeric character with only a negative sign overpunched
   CR or DB represents the actual two positions which are attached if the field is negative; two blanks are attached if the field is positive
   B represents a position which is always blank
   0 represents a position which is always zero

   $ represents a position which may contain an actual dollar sign. (See note on Editing)
   . represents a position which may contain an actual decimal point. (See note on Editing)
   , represents a position which may contain an actual comma. (See note on Editing)

J, K, L, M, N each represent a special character whose meaning will be determined by each implementor.

The specific set of characters (n) where n is any integer may be used following any of the above characters. This usage will specify that the class of data represented by the character immediately preceding the left parenthesis exists in the physical data n times. (e.g., A(3)X(2) and AAAXX are equivalent.)

FUNCTION: To define the assumed decimal or binary point.

$$\left[\text{\underline{POINT} LOCATION IS} \left\{ \genfrac{}{}{0pt}{}{\text{\underline{LEFT}}}{\text{\underline{RIGHT}}} \right\} \text{integer} \left\{ \genfrac{}{}{0pt}{}{\text{PLACES}}{\text{\underline{BITS}}} \right\} \right]$$

Notes:

1. When the field PICTURE is not given, the definition of numeric fields having non-integral values requires the use of this clause.

2. An actual decimal point (i. e., the character ". ") may not be defined through the use of this clause. Actual decimal points must be shown in the field PICTURE.

3. BITS must be specified in the definition of binary points. Absence of the word BITS indicates an assumed decimal position.

4. The point is located "integer" positions to the left or right of the least significant position of the field. COBOL implementors need not accept the BITS option in compilers for decimal computers.

RANGE

FUNCTION: To supply additional information which may aid the compiler in optimizing the object program.

$$\left[\underline{\text{RANGE}} \text{ IS literal-1 } \underline{\text{THRU}} \text{ literal-2} \right]$$

Notes:

1. The minimum and maximum value of the data being described are given by "literal-1" and "literal-2" respectively.

REDEFINES

FUNCTION: To allow the same memory area to have more than one name and description.

$$\Big[\ \underline{REDEFINES}\ \text{data-name}\ \Big]$$

Notes:

1. When used, this clause must immediately follow the data-name of this entry.

2. The area to be redefined starts at the level of "data-name" and ends when the same or a lower level number is encountered. The new description of the area begins with the entry containing the REDEFINES. The level number of this entry must be the same as the level number of "data-name".

3. The entries giving the new description of the area must immediately follow the entries describing the area being redefined, except when redefinition occurs at level number 1.

4. This clause is not used for logical records associated with the same file. These records must be listed in the File Description entry and will automatically share the same memory area.

FUNCTION: To specify the presence of a sign which is not shown in the field picture.

$$\left[\left\{\begin{array}{l}\underline{\text{SIGNED}}\\ \underline{\text{SIGN}} \quad \text{IS data-name}\end{array}\right\}\right]$$

Notes:

1. This clause is required when there is a sign which must be associated with the field being defined and when that sign is not shown in the field PICTURE.

2. The SIGNED option implies that the sign of the field appears in the sign position associated with the most significant position of the field. SIGNED is used when:

   a) The sign is unmappable and it appears in the "standard sign position".

   b) The sign is in a standard mappable sign position but the sign position has not been mapped. Such a case may only exist when the field does not share a computer word with any other field. In addition to the above requirements, the character representing the sign must be expressed in the same BASE as the characters comprising the field.

3. The SIGN IS option is used when the sign of the field has been defined as a separate field. The function of this option is to provide a crossreference between the field and its sign. SIGN IS must be used in the following instances:

   a) When the sign is mappable but applies to more than one field.

   b) When the sign is mappable but physically separated from the field by the intervention of one or more fields.

   c) When the BASE of the character representing the mappable sign differs from that of the characters comprising the field. This concept is consistent with the rule that all characters in a field must have a homogeneous BASE.

4. Non-standard unmappable signs can not be defined. They can not be defined in the sign clause, nor can they be shown in the field PICTURE.

5. All other signs must be shown in the field PICTURE.

SIZE

FUNCTION: To specify, in terms of the number of characters, the size of the data currently being defined.

$$\left[\underline{SIZE} \text{ IS } \left[\text{integer-1 } \underline{TO} \right] \text{ integer-2 CHARACTERS} \right. \\ \left. \left[\underline{DEPENDING} \text{ ON field-name} \right] \right]$$

Notes:

1. The SIZE clause is only required at the lowest level of data being described. SIZE can only be specified in terms of homogeneous characters (i.e., characters represented in the same BASE). For this reason, SIZE may not be specified at levels which contain characters having different bases.

2. If only "integer-2" is shown, it represents the exact number of characters. If "integer-1" and "integer-2" are both shown, they refer to the minimum and maximum number of characters respectively. If "integer-1" is zero, the data being described will not necessarily be present.

3. The DEPENDING option pertains to variable length data. It is only required when the means of determining the end of variable length data is not otherwise defined. Thus, data ending with a special hardware symbol, which is shown in the PICTURE by J, K, L, M, or N, does not require the use of the DEPENDING option.

4. The use of "field-name" means that the count of the number of characters in the data is contained within the data unit called "field-name". This value must appear in each record and be an integer. "field-name" must appear within the record to which the current Record Description entry pertains. Furthermore, the value of "field-name" must either be available at object time prior to giving the READ, or must appear in the record being read. If it appears in the record, it must be in that portion which is always present and must precede any variable portion.

SYNCHRONIZED

FUNCTION: To specify positioning of a field within a computer word or words.

$$\left[\underline{\text{SYNCHRONIZED}} \left\{ \begin{array}{l} \underline{\text{LEFT}} \\ \underline{\text{RIGHT}} \end{array} \right\} \right]$$

Notes:

1. This clause applies to fixed word length computers, and aids in maintaining compatibility across computers with different word structures.

2. When used, this clause indicates that the field is positioned as specified within one or more computer words, and that no other pertinent information is contained in these words.

3. When LEFT is specified, the least significant portion of the (last) word contains zeros or spaces as defined by Note 4. When RIGHT is specified, the most significant portion of the (first) word contains zeroes or spaces as defined by Note 4.

4. If the field being SYNCHRONIZED contains alphabetic or alphanumeric characters, spaces appear in the unused part of the word. If the field contains numeric characters, zeroes appear in the unused part of the word. Zeroes and spaces are represented in the same BASE as the field itself.

5. When SYNCHRONIZED is used in conjunction with SIGNED, if the "computer sign position" is mappable, the sign of the field appears in the normal "computer sign position" of the first word regardless of whether the field is SYNCHRONIZED LEFT or RIGHT.

VALUE

FUNCTION: To define the value of constants, the initial value of working storages, or the value associated with a condition-name.

$$\left[\text{ } \underline{\text{VALUE}} \text{ IS literal } \right]$$

Notes:

1. When used in a condition-name entry, no further information is required.

3.2.3 Specific Entry for Condition-Name

Each condition-name requires a separate entry with the level number 88. This entry contains only the name of the condition and the value of the condition. The condition-name entries for a particular conditional variable must follow the entry describing the field with which the condition-name is associated.

More specifically,

nn field-name . . .

88 condition-name-1 VALUE IS literal-1.

88 condition-name-2 VALUE IS literal-2.

.
.
.

As an example,

3 GRADE . . .

88 FRESHMAN VALUE IS 1.

88 SOPHOMORE VALUE IS 2.

88 JUNIOR VALUE IS 3.

88 SENIOR VALUE IS 4.

4. SUMMARY

4.1 FILE SECTION

4.1.1 Organization

The FILE SECTION contains a Section header, File Description entries, Record Description entries for label records and Record Description entries for data records. Some of the information about the file may be in the COBOL library and therefore may not appear explicitly in the FILE Section. The order of information is as follows:

FILE SECTION.

FD file-name . . .

01 label-name . . .

.
.
.

```
01 record-name
      .
      .
      .

FD file-name . . .
      .
      .
      .
```

4.1.2 Specifications and Handling of Labels

The COBOL System provides for the automatic handling of four types of labels – beginning and ending, file and tape. The notes for the LABEL RECORDs clause in the File Description entry contain an indication of the relative position of these labels on the tape(s) associated with a file.

A label record is a logical record containing the labelling information about a tape or file. There are many different types of label records, but some of these are fairly standard. In order to have common recognition of these records, fixed names have been assigned. Any other label records and the methods of handling them will be specified by the individual implementors. The label records with fixed names are:

BEGINNING-TAPE-LABEL

BEGINNING-FILE-LABEL

ENDING-FILE-LABEL

ENDING-TAPE-LABEL

A Record Description must be available for each label record employed. Record Descriptions for label records are prepared in the same manner as those prepared for data records; however, since the input-output system must perform special operations on certain fields within the label record, fixed names have been assigned to those label fields which have particular significance.

For purposes of discussion, label fields are classified as follows:

a) Those which must contain unique values depending on the particular file involved (e.g., name and number of the file, etc.)

b) Those which contain hash-totals.

c) Those which have a general meaning in the processing of files (e.g., record count, tape number, etc.)

d) Those having special functions which are not handled automatically.

A field which must have a unique value depending on the particular file is handled in the following manner:

a) In preparing the Record Description entry, any name may be assigned to the field. Since the value of the field is a variable, it is not shown.

b) In preparing the File Description entry, the name of the field and the value which it must contain is listed in the VALUE clause.

c) In processing an input file, an equality test is made between the contents of the label field, and the corresponding value specified in the file description.

d) In processing an output file, the value specified in the file description is entered in the label field of the beginning label.

A hash-total field is handled as follows:

a) In preparing the Record Description entry, any name may be assigned to the field.

b) In preparing a File Description entry, the name of the label field is listed with the corresponding name of the data field whose "hashed value" is contained in the field. See HASHED option of VALUE clause in File Description Entry.

c) In processing an input file, an automatic hash-total will be accumulated from the data field named. This total will then be compared against that found in the ending label.

d) In processing an output file, an automatic hash-total will be accumulated from the data-field named. This total will then be entered in the label-field of the ending label.

Label records contain within themselves label fields. Again, there are many different label fields, but some of these are fairly standard. In order to have common recognition of these fields, fixed names have been assigned. Any other label fields and the method of handling them will be specified by the individual implementors. The label fields with fixed names are:

REEL-NUMBER

DATE-WRITTEN

PURGE-DATE

TEST-PATTERN

BLOCK-COUNT

RECORD-COUNT

MEMORY-DUMP-KEY

SENTINEL

The last two names in the above list represent conditional variables for which the following fixed condition-names must be specified:

MEMORY-DUMP   } are condition names for the con-
NO-MEMORY-DUMP } ditional variable MEMORY-DUMP-
                            KEY

END-OF-FILE   } are condition names for the con-
END-OF-TAPE } ditional variable SENTINEL

All other label fields (i. e., class "d" fields), must be handled by the programmer. That is, the processing required for such fields must be specified in the PROCEDURE DIVISION of the program. Since the time at which such statements must be executed is under the control of the input-output system, the programmer must identify those COBOL statements which are to be executed during the preparation or checking of particular labels by the input-output system. This is accomplished through the USE verb.

4. 2 WORKING STORAGE SECTION

4. 2. 1 Organization

The WORKING-STORAGE Section contains a Section header, Record Description entries for working storage records and Record Description entries for non-contiguous working storage fields. The order of the information which may appear is as follows:

WORKING-STORAGE SECTION

    77 field-name . . .

        .

        .

        . .

    77 field-name . . .
    01 record-name . . .

        .

        .

        .

    01 record-name . . .

        .

        .

        .

4. 2. 2 Non-Contiguous Working-Storages

Working storage fields which bear no relationship to one another need not be grouped into records providing they do not need to be further subdivided. Instead, they may be classified and defined as non-contiguous fields. Such fields are defined in separate Record Description entries which begin with the special level number 77.

4.2.3 Initial Values

Initial values of working storages may be specified in the VALUE clause of the Record Description Entry.

4.2.4 Condition-names

As with data fields, a working storage field may constitute a conditional variable to which one or more condition-names may be associated. Entries defining condition-names must immediately follow the field to which they relate. Condition-names may be associated with non-contiguous as well as contiguous working storage fields.

4.3 CONSTANT SECTION

4.3.1 Organization

The CONSTANT Section contains a section header, Record Description entries for constant records (i. e., groups of constants, such as a table, whose relative positions are significant), and Record Description entries for non-contiguous constants. The order of the information which may appear is as follows:

```
            CONSTANT SECTION

       77  field-name . . .
              .
              .
              .
       77  field-name . . .
       01  record-name . . .
              .
              .
              .
       01  record-name . . .
              .
              .
              .
```

4.3.2 Description of Constants

The minimum amount of information required in the definition of a non-contiguous constant is the level number, constant name, and the value of the constant. Since contiguous (grouped) constants may be addressed by use of subscripts, an internal map is necessary and BASE is required in addition to the minimum information required for definition of a non-contiguous constant.

4.3.3 Tables of Constants

Tables of constants must be defined in the following manner:

a) The table is considered to be a record, and, therefore, is defined by a set of contiguous record description entries which define the contents of the table.

b) The record must then be redefined (see the REDEFINES clause in the Record Description entry), to show the hierarchal structure inherent in the table. The OCCURS clause must be used in the redefinition, if subscripts are to be employed in referencing the table. In redefining a table, complete Record Description entries are required (i. e., SIZE, BASE, PICTURE, editing information, etc.).

5. LIST OF KEY AND OPTIONAL WORDS IN THE DATA DIVISION

5.1 KEY WORDS

The key words in the DATA DIVISION are:

ALPHABETIC	LABEL
ALPHANUMERIC	LEAVING
AN	LEFT
BASE	LIBRARY
BASIC	MEMORY-DUMP
BEGINNING-FILE-LABEL	MEMORY-DUMP-KEY
BEGINNING-TAPE-LABEL	NO-MEMORY-DUMP
BITS	NUMERIC
BLANK	OCCURS
BLOCK	OMITTED
BLOCK-COUNT	OTHER
CHECK	PICTURE
CLASS	POINT
CONSTANT	PREPARED
COPY	PURGE-DATE
DATA	RANGE
DATA-WRITTEN	RECORD
DEPENDING	RECORD-COUNT
DIVISION	RECORDING
END-OF-FILE	REDEFINES
END-OF-TAPE	REEL-NUMBER
ENDING-FILE-LABEL	RIGHT
ENDING-TAPE-LABEL	SECTION
FD	SENTINEL
FILE	SEQUENCED
FILLER	SIGN
FLOAT	SIGNED
HASHED	SIZE
JUSTIFIED	STANDARD

SUPPRESS	TO
SYNCHRONIZED	WORKING-STORAGE
TEST-PATTERN	VALUE
THRU	ZERO

5.2 OPTIONAL WORDS

The optional words in the DATA DIVISION are:

ABOUT
ARE
CHARACTERS
CONTAINS
DOLLAR
FOR
IN
IS
LOCATION
MODE
OF
ON
PLACES
PROTECTION
TIMES
TYPE
WHEN

VII. ENVIRONMENT DIVISION

1. GENERAL DESCRIPTION

1.1 OVERALL APPROACH

The basic approach to the ENVIRONMENT DIVISION is to centralize those aspects of the total data processing problem which are dependent upon the physical characteristics of a specific computer. It provides a linkage between the logical concepts of data and records, and the physical aspects of the files on which they are stored.

The ENVIRONMENT DIVISION is the one part of the COBOL system which must be rewritten each time a given problem is run on a different computer. It has been included in the COBOL System to provide a standard way of expressing the computer dependent information which must be included as part of every problem.

1.2 ORGANIZATION

The ENVIRONMENT DIVISION has been divided into two sections - CONFIGURATION and INPUT-OUTPUT.

The CONFIGURATION Section, which deals with the overall specifications of computers, is divided into three paragraphs. They are: the SOURCE-COMPUTER, which defines the computer on which the COBOL Compiler is to be run; the OBJECT-COMPUTER, which defines the computer on which the program produced by the COBOL Compiler is to be run; and SPECIAL-NAMES, which relate the actual names of the hardware used by the COBOL Compiler to the names used in the program.

The INPUT-OUTPUT Section deals with the definition of the external media and information needed to create the most efficient transmission and handling of data between the media and the object program. This section is divided into two paragraphs. They are the I-O Control, which defines special input-output techniques, rerun, and multiple file tapes; and FILE-CONTROL, which names and associates the files with the external media.

1.3 STRUCTURE

The following is a general outline of the Sections and Paragraphs under the ENVIRONMENT DIVISION.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER.   computer name . . .

OBJECT-COMPUTER.   computer name . . .

SPECIAL-NAMES.   hardware-name IS . . .

INPUT-OUTPUT SECTION.

FILE-CONTROL.   SELECT . . .

I-O-CONTROL.   APPLY . . .

2. CONFIGURATION SECTION

  2.1 SOURCE COMPUTER

FUNCTION: To describe the computer upon which the program is to be compiled.

  To provide a means of communicating with an executive routine.

Option 1:

  SOURCE-COMPUTER. COPY library-name.

Option 2:

  SOURCE-COMPUTER. Computer-name $\left[\text{WITH } \underline{\text{SUPERVISOR}} \text{ CONTROL} \right]$

$$\left[\text{, } \underline{\text{MEMORY}} \text{ SIZE} \left\{ \begin{array}{l} \text{integer-1} \left\{ \begin{array}{l} \underline{\text{WORDS}} \\ \underline{\text{CHARACTERS}} \\ \underline{\text{MODULES}} \end{array} \right\} \\ \underline{\text{ADDRESS}} \text{ integer-2 } \underline{\text{THRU}} \text{ integer-3} \\ \qquad \left[\text{, integer-4 } \underline{\text{THRU}} \text{ integer-5 } \ldots \right] \end{array} \right\} \right]$$

$$\left[\text{, } \left[\text{integer-6} \right] \text{ hardware-name-1 } \left[\text{, } \left[\text{integer-7} \right] \text{ hardware-name-2 } \ldots \right] \right].$$

Notes:

  1. Option 1 is used when the COBOL library contains the entire description of the source computer.

  2. Fixed "computer-name"s and "hardware-name"s will be assigned by the individual implementors.

  3. The "computer-name" provides an automatic definition of a particular equipment configuration. The "computer-name" and its implied configuration is specified by the implementor. The configuration definition contains specific information concerning the memory size, memory addresses, and types and number of hardware for a specific computer.

  4. The configuration defined by "computer-name" may comprise more equipment than is actually needed by the compiler. In that case, Option 2 allows the user to specify the actual subset of the configuration he wishes to use. The compiler replaces the number associated with the hardware in the "computer-name" configuration with the specified "integer".

SOURCE-COMPUTER

5. If the subset specified by the user is less than the minimal configuration required for compilation, an error will be indicated.

6. "Hardware-name"s may include input/output units, floating point hardware, indicators (i.e., breakpoints, sense devices), index registers, any special or additional instructions, etc.

7. The SUPERVISOR option is used when it is planned that the compiler will be run under control of an executive routine. Communication between the compiler and the executive routine will be specified by each implementor.

2.2 OBJECT COMPUTER

FUNCTION: To describe the computer upon which the program is to be run.

To provide a means of communicating with an executive routine.

Option 1:

OBJECT-COMPUTER. COPY library-name.

Option 2:

OBJECT-COMPUTER. Computer-name $\left[\text{WITH } \underline{\text{SUPERVISOR}} \text{ CONTROL} \right]$

$\left[, \underline{\text{MEMORY}} \text{ SIZE } \left\{ \begin{array}{l} \text{integer-1} \left\{ \begin{array}{l} \underline{\text{WORDS}} \\ \underline{\text{CHARACTERS}} \\ \underline{\text{MODULES}} \end{array} \right\} \\ \underline{\text{ADDRESS}} \text{ integer-2 } \underline{\text{THRU}} \text{ integer-3} \\ \qquad \left[, \text{ integer-4 } \underline{\text{THRU}} \text{ integer-5 . . .} \right] \end{array} \right\} \right]$

$\left[, \left[\text{integer-6} \right] \text{hardware-name-1} \left[, \left[\text{integer-7} \right] \text{hardware-name-2 . . .} \right] \right]$.

Notes:

1. Option 1 is used when the COBOL library contains the entire description of the object computer.

2. Fixed "computer-name"s and "hardware-name"s will be assigned by the individual implementors.

3. The "computer-name" provides an automatic definition of a particular equipment configuration. The "computer-name" and its implied configuration is specified by the implementor. The configuration definition contains specific information concerning the memory size, memory addresses, and types and number of hardware for a specific computer.

4. The configuration defined by "computer-name" may comprise more equipment than is actually needed by the object program or is available on the object computer. In that case, Option 2 allows the user to specify the actual subset of the configuration he wishes to use. The compiler replaces the number associated with the hardware in the "computer-name" configuration with the specified "integer".

5. If the subset specified by the user is less than the minimal configuration required for the program to run, an error will be indicated during compilation.

OBJECT-COMPUTER

6. "Hardware-name"s may include input-output units, floating point hardware, indicators (i.e., breakpoints, sensing devices), index registers, any special or additional machine instructions, etc.

7. The SUPERVISOR option is used when it is planned that the object program will be run under the control of an executive routine. Communication between the object program and the executive routine will be specified by the individual implementor.

2.3 SPECIAL-NAMES

FUNCTION: To provide a means of relating hardware with mnemonic-names and the status of hardware switches with condition-names:

Option 1:

SPECIAL-NAMES. COPY library-name.

Option 2:

SPECIAL-NAMES. Hardware-name-1 IS mnemonic-name-1

$\left[\text{, hardware-name-2 IS mnemonic-name-2 . . .} \right]$.

Option 3:

SPECIAL-NAMES. Hardware-name-1 $\left[\text{IS mnemonic-name-1} \right]$

$\left[\text{, ON STATUS IS condition-name-1} \right]$ $\left[\text{, OFF STATUS IS} \right.$

$\left. \text{condition-name-2} \right]$ $\left[\text{, hardware-name-2 . . .} \right]$.

Notes:

1. This paragraph is not required if "mnemonic-names" and "condition-names" are not used in the PROCEDURE DIVISION.

2. Option 1 is used when the COBOL library contains the entire description of the special-names used in the program.

3. In Option 2, "hardware-name" may not be a switch. Each "hardware-name" may have assigned to it a "mnemonic-name" which may then be used in the ACCEPT and DISPLAY verbs.

4. In Option 3, "hardware-name" must be a switch and there must be associated with it either a "mnemonic-name" or a "condition-name" or both. Each "hardware-name" may have assigned to it a "mnemonic-name" which may then be used in the ACCEPT and DISPLAY verbs. The status of the switches is specified by using "condition-name"s and interrogated by testing the "condition-name"s. (See Conditional Variables in Chapter V).

5. Fixed "hardware-name"s are assigned by the individual implementor.

6. Combinations of Option 2 and 3 may be used.

3. INPUT-OUTPUT SECTION

3.1 FILE-CONTROL

FUNCTION: To name each file, identify its media and allow particular hardware assignments. To specify alternate input-output areas. To facilitate multiprogramming.

Option 1:

FILE-CONTROL. COPY library-name.

Option 2:

FILE-CONTROL. SELECT [OPTIONAL] file-name-1 [RENAMING file-name-2] , ASSIGN TO [integer-1] hardware-name-1 [, hardware-name-2 ...]

[FOR MULTIPLE REEL] [, RESERVE $\begin{Bmatrix} \text{integer-2} \\ \underline{\text{NO}} \end{Bmatrix}$ ALTERNATE $\begin{Bmatrix} \text{AREA} \\ \text{AREAS} \end{Bmatrix}$]

[, PRIORITY IS priority] . [SELECT ...] .

Notes:

1. Option 1 is used when the COBOL library contains the entire description of the FILE-CONTROL.

2. The beginning of the information for each "file-name-1" will be identified by the key word SELECT.

3. The name of each selected file (e.g. "file-name-1") must be unique within a program.

4. The key word OPTIONAL is required for input files which will not necessarily be present each time the object program is to be run.

5. If more than one file utilizes the same FILE DESCRIPTION, the RENAMING option must be included. That is, "file-name-1" utilizes the FILE DESCRIPTION written for "file-name-2", i.e., when a file is to be processed both as an input and output in the same program. RENAMING "file-name-1" or "file-name-2" implies the sharing of a single FILE DESCRIPTION and does not allow these files to be referenced interchangeably in the program.

6. All files used in the program must be assigned to an input or output medium ("hardware-name"). "Hardware-name"s are specified by the implementors. These names may be broadened by the implementors to include additional information about the media.

FILE-CONTROL

As an example, for card equipment, the "hardware-name" may include stacker selection; for drums, "hardware name"s may include drum addresses or a storage selection technique.

7. "Integer-1" may only be specified when "hardware-name-1" is tape. "Integer-1" then indicates the number of tape units to be assigned to the file.

8. If "integer-1" is not specified (in the case of tape) the compiler will then determine the exact number of units to be assigned. The number of available units is specified in the OBJECT-COMPUTER paragraph.

9. The MULTIPLE REEL option must be included when:

    (a) "Integer-1" is not specified and more than one reel may exist in a file.

    (b) "Integer-1" is specified but may be less than the total number of reels in a file.

10. Specific hardware units are assigned by using specific "hardware-name"s.

11. When specific input-output units are assigned by the user, the same unit must be assigned to all files existing on the same reel (see MULTIPLE FILE option in I-O-CONTROL paragraph.)

12. The RESERVE statement allows the user to modify the standard input-output alternate areas allocated by the compiler.

13. The PRIORITY option provides a means of assigning priorities to individual files for multiprogramming operations. The manner in which the priority is specified will be defined by the implementor.

3.2 I-O-CONTROL

FUNCTION: To specify the input-output techniques, the points at which rerun is to
be established, the memory area which is to be shared by different
files, and the location of files on a multiple file reel.

Option 1:

I-O-Control. COPY library-name.

Option 2:

I-O-Control. $\left[\underline{\text{APPLY}} \text{ input-output techniques.} \right]$

$\left[\underline{\text{RERUN}} \left[\underline{\text{ON}} \left\{ \begin{matrix} \text{file-name-1} \\ \text{hardware-name} \end{matrix} \right\} \right] \text{ EVERY} \right.$

$\left\{ \begin{matrix} \left\{ \begin{matrix} \text{END OF } \underline{\text{REEL}} \\ \text{integer-1 } \underline{\text{RECORDS}} \end{matrix} \right\} \text{ OF file-name-2} \\ \text{integer-2 } \underline{\text{CLOCK-UNITS}} \\ \text{condition-name} \end{matrix} \right\} \cdot \left. \right]$

$\left[\underline{\text{SAME}} \text{ AREA FOR file-name-3 } \left[, \text{ file-name-4 } \ldots \right] . \right]$

$\left[\underline{\text{MULTIPLE FILE}} \text{ TAPE CONTAINS file-name-5 } \left[\underline{\text{POSITION}} \right. \right.$
$\text{integer-3}\left]\left[, \text{ file-name-6 } \left[\text{ POSITION integer-4} \right] \ldots \right] . \left. \right]$

Notes:

1. This paragraph is required only when one of the above options is desired.

2. Option 1 is used when the COBOL library contains the entire description of
I-O-CONTROL.

3. It is assumed that some implementors will furnish more than one input-
output system technique. The "input-output technique" option allows the
user to select the appropriate technique for his object program.

4. If RERUN is specified, it is necessary to indicate when a rerun point is to be
established and where the memory dump is to be written.

    a) Memory dumps are written in the following ways:

        i) The memory dump is written on each reel of an output file. Each
implementor will specify where the memory dump is to be written
on the reel.

ii) The memory dump is written on a separate rerun tape ("hardware-name").

b) Many methods of establishing rerun points are available. Rerun points are established by the following conditions:

i) When the end of REEL option is used and it is also desired to write the memory dump on an output file ("file-name-2"). In this case, "file-name-1" is not required. For example,

RERUN EVERY END OF REEL OF UPDATED-INVENTORY.

ii) If "file-name-1" (which must be an output file) is specified for the RERUN, normal reel closing functions for "file-name-1" will be performed along with the memory dump. In this case, "file-name-2" may either be an input or output file.

iii) When a number of records ("integer-1") of an input or output file ("file-name-2") have been processed. In this case, "hardware-name" must be specified.

iv) When an interval of time ("integer-2") calculated by an internal clock, has lapsed. In this case, "hardware-name" must be specified.

v) When a hardware switch assumes a particular status ("condition-name"). In this case, "hardware-name" must be specified. The "condition-name" and the associated hardware switch must be defined in the SPECIAL-NAMES paragraph of the CONFIGURATION SECTION.

5. The function of SAME AREA under Option 2 is to provide for the overlaying of different files on the same memory area. (Area includes the record area as well as the alternate storage area). When one file is to overlay another, the first file must be "closed" before the second file can be "opened".

6. The MULTIPLE FILE option is required when more than one file shares the same physical reel of tape. Regardless of the number of files on a single tape, only those files which are used in the object program need be specified. If all file-names have been listed in consecutive order, the POSITION need not be given. If any file in the sequence is not listed, the position relative to the beginning of the tape must be given.

4. LIST OF KEY AND OPTIONAL WORDS IN THE ENVIRONMENT DIVISION

4.1 KEY WORDS

The key words in the ENVIRONMENT DIVISION are:

ADDRESS	OBJECT-COMPUTER
APPLY	OFF
ASSIGN	ON
CHARACTERS	OPTIONAL
CLOCK-UNITS	POSITION
CONFIGURATION	PRIORITY
COPY	RECORDS
ENVIRONMENT	REEL
DIVISION	RENAMING
FILE	RERUN
FILE-CONTROL	RESERVE
I-O-CONTROL	SAME
INPUT-OUTPUT	SECTION
IS	SELECT
MEMORY	SOURCE-COMPUTER
MODULES	SPECIAL-NAMES
MULTIPLE	SUPERVISOR
NO	THRU
	WORDS

4.2 OPTIONAL WORDS

The optional words for the ENVIRONMENT DIVISION are:

ALTERNATE

AREA

AREAS

CONTAINS

CONTROL

END

EVERY

FOR

OF

SIZE

STATUS

TAPE

TO

WITH

VIII. REFERENCE FORMAT

1. GENERAL DESCRIPTION

The purpose of the Reference Format is to provide a standard way of writing COBOL programs. This Reference Format is an output from the compilation and will be used as a means of communication, and as one of the acceptable input formats for each COBOL compiler. Because the language allows the use of sentences, paragraphs, sections, and divisions, a standard method of representing these will be shown. The basic principle behind the particular formats chosen is to allow the maximum amount of flexibility for individual tastes while still using one form.

It is essential to note that whenever the Reference Format is being used, its specifications take precedence over any other rules allowing an arbitrary number of spaces. Thus, the number of spaces is critical in the Reference Format.

There are three parts to the Reference Format, one for each division.

2. PROCEDURE DIVISION

The specific format for the PROCEDURE DIVISION is as follows:

```
Seq. No.    A    B    C                                    Z
   (6)     (1)  (4)  (4)
```

The first column in the Reference Format occupies 6 spaces and is used for the sequence number. One blank space is left and then the remainder of the page is used for the text and names of parts of the text. Positions A, B, and C have special significance to be described below. The right hand margin Z is determined by the width of the paper or printing media.

The names of the division, and names of the sections and paragraphs within it start under position A. Position B is the normal left hand margin for the text, and C is used for indentation purposes.

The first line contains the header -

PROCEDURE DIVISION.

Section names may be used, if desired. A section name applies to all succeeding information until another section name is reached. The section name is followed by a space, the word SECTION, a period and a space. The only other information which may appear on the same line is the priority number described in IX. 2, "Segmentation". A section name may be used as a qualifier for otherwise identical paragraph names. Within a section, any reference to a paragraph name not otherwise qualified will be assumed to refer to paragraphs within the section.

A paragraph may be either named or unnamed. When it is named, the name is followed immediately by a period and a space. The first sentence of the paragraph may begin anywhere on the same line as the name, or under position B on the next line. A new paragraph is determined by either the appearance of another paragraph name, or the beginning of a sentence under or to the right of position C

on a line not containing a paragraph name. (This indentation is important because the PERFORM verb may indicate a range of execution of paragraphs, and the scope of the verb will terminate when a new paragraph is reached.) A paragraph may consist of only one sentence.

Any sentence which occupies more than one line must be continued by starting under B on the next line, or else the continuation may be mistaken for another paragraph.

If a word or literal must be split between two lines, this must be indicated by having a hyphen in the last character position on the first line.

Thus, a hyphen appearing in the last character position on a line is never considered part of the word or the literal. Because words may not end with a hyphen, and a minus sign is surrounded by spaces, there is no ambiguity. If the character in the word or literal which would normally be in the last position, happens also to be a hyphen, it will appear as the first character on the second line. Thus, there will actually be two hyphens printed - one at the end of the first line and one at the beginning of the second. Only the second is part of the word or literal.

No punctuation mark other than the hyphen, quote or left parenthesis should be in the first position of a line, unless it is within a literal.

SAMPLE

Seq.   A   B   C   Z
No.
```
        |       |   |                                    |
        |       |   |                                    |
        |       |   |                                    |
            PROCEDURE DIVISION.

            yyyyyyyy SECTION.

            paragraph-name.  aaaaaaaaaaaa.  bbbbbbbbbbbbbbbbbbbbbbbbbbb

                bbbbbbbbbbbbbbbbbbbbb.  cccccccccccc.  dddddd

                ddddddddddddd.  eeeeeeeeeeeeeee.

                    ffffffffffffffffffffffffffffff.  gggggggggggggg

                ggggggggggg.  hhhhhhhhhhhhhhhhhhhhh.  iiiiiii

                iiiiiiiiiii.

            par-name.  jjjjjjjjjjjjjjjjjjjjjjjjjjj.

            par-name.  kkkkkkkkkkk.
```

Note: If the user wishes to have the verb stand out, then he should use only simple statements each being a full paragraph, and short paragraph names.

3. DATA DIVISION

The basic unit in the Record Description is an entry. This entry is started by a level number and followed by the name of the data and a series of independent clauses. The clauses may be preceded by a semicolon and the entry is ended by a period followed by a space. This applies to both the File Description and the Record Description.

The basic difference between this format and that of the procedure description is the concept of level number. The latter defines a very detailed hierarchal structure and it is desirable to show this in the Reference Format. This is done by allowing the indentation of the level number and the corresponding entry.

The first line consists of the header DATA DIVISION $\left[\text{PREPARED FOR} \right]$. The sequence number appears at the left as in the Reference Format for the PROCEDURE DIVISION. The first level number then starts at the place where the procedure names are placed. If a single entry requires more than one line, the left margin for each is the same, namely, the position under the first letter of the data-name. The rules for breaking words at the end of a line are the same as in the PROCEDURE DIVISION. Each new level number is indented 4 spaces further than the previous one. (If this causes the right hand margin to be reached, then all subsequent lower levels are not indented any further.) Two spaces are left between the level and the data-name. The FD (ie. File Description) entry always begins at the left margin of the text.

SAMPLE:

nn data-name-1 yyyyyyyyyyyyyyyyyyyy; zzzzzzzzzzzz; ssssssssssssssss

sssssssssss.

    nn data-name-2 aaaaaaaaa; bbbbbbbbbbbbbbbbbbb; ccccccccccccccccc

        ccccccccccccccccccccccccccccccc.

    nn data-name-3 dddddddddddddddddddddddddd.

nn data-name-4 eeeeeeeeeeeeeeeeeeee; ffffffffffffffff; gggggggggg

ggggggggggggggggggggggggggggggggggg.

    nn data-name-5 hhhhhhhhhhhhhhhhhhhh.

        nn data-name-6 jjjjjjjjjjjjjjjjjjjjjjjjjjjj.

        nn data-name-7 kkkkkkkkkkkkkkkkkkkkk.

4. ENVIRONMENT DIVISION

The Reference Format for the ENVIRONMENT DIVISION is the same as that of the PROCEDURE DIVISION. However, there are only fixed section and paragraph names, and no unnamed paragraphs. The rules for continuation of sentences and words are the same. The I-O-CONTROL and the FILE-CONTROL

paragraphs are each composed of several sentences, whereas the other paragraphs are composed of only one sentence each. The following is an example of a possible format, except that all the information for each section and paragraph has not been shown, (See Chapter VII, ENVIRONMENT DIVISION).

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. computer-name .. .　．

OBJECT-COMPUTER. computer-name .. .　．

SPECIAL NAMES. hardware-name IS　...　。

INPUT-OUTPUT SECTION.

FILE-CONTROL. SELECT file-name-1...　.　SELECT

file-name-2 ... .　SELECT file-name-n ... .

I-O-CONTROL. APPLY ... . SAME ... . MULTIPLE ... .

IX. SPECIAL FEATURES

1. LIBRARIES

The COBOL library contains three types of entries, corresponding to the three divisions of the COBOL system. Thus, there is information describing machine configurations which is retrievable through the use of the COPY in the ENVIRON-MENT DIVISION, and file and record descriptions retrievable through the use of the COPY in the DATA DIVISION. Finally there are procedure statements – commonly called subroutines – which are retrievable through the use of the verb INCLUDE in the PROCEDURE DIVISION. Each division is capable of obtaining material pertaining only to itself.

The actual physical makeup of the COBOL library, as well as the maintenance and handling, are left to the individual implementor.

The calling of library material produces the same effect as if the programmer had written the same material in his source program. Because information from the ENVIRONMENT and DATA DIVISION is never directly translated to object coding, library material from these divisions is obtained at compilation time and then used as if it had originally been part of the source program. Because the procedure statements in the source code are translated to form the object program, the library material from the PROCEDURE DIVISION is brought in at compilation time for use as part of the object program. However, it is almost always necessary for the compiler to make modifications and/or additions to the subroutines in order to provide correct and efficient linkage with the main program.

Subroutines in the library are identified by a library name, and may have any number of paragraph names, but may not have any section name. The subroutines are called by name from the COBOL library by the compiler directing verb IN-CLUDE, and at the option of the programmer are inserted either at the place the INCLUDE occurs, or outside of the normal program sequence. When necessary, the compiler will supply as a qualifier either the name of the paragraph containing the INCLUDE, or an arbitrary name generated during compilation. REPLACING, if specified by the INCLUDE verb, is accomplished at the time of compilation. Once inserted into the program, subroutines may be referenced either by a PER-FORM or a GO, or may simply be executed in line without any special reference.

2. SEGMENTATION

Segmentation is the partitioning of the object program into memory loads, in those cases where the memory is too small to contain all the object coding. The COBOL system does not provide segmentation of data. SECTIONS in the PROCEDURE DIVISION enable the user to indicate the points at which the object program may be segmented and thus permit the compiler to provide semi-automatic segmentation when necessary. (Fully automatic segmentation occurs only when the compiler itself determines the size of each memory load without indication by the user. Semi-automatic segmentation occurs when the compiler uses programmer supplied information to determine memory loads and provides for automatic reloading where necessary.)

Segment points are defined by sections. That is, each SECTION in the PRO-CEDURE DIVISION is automatically considered as a possible memory load. Further, the word SECTION may be followed immediately by a priority number that

will instruct the compiler which segment points to try first. The priority number must be a decimal integer between 1 and 999 and may not be a field name. If a priority number is used, the compiler will attempt to segment first at the lowest number and successively try with higher numbers until the program is reduced to a series of acceptable memory loads.

In order to preserve loops intact, it will be essential to insert segment points in pairs surrounding the loop, using the same priority number for each SECTION in the pair, and using the priority number nowhere else.

As an illustration, consider the following layout:

A SECTION. 1

B SECTION. 99

C SECTION. 99

D SECTION.

E SECTION. 98

F SECTION.. 98

G SECTION. 2

In this case, Sections A and G are most preferable to cut off, B and C are closely related and are not to be separated, E and F are likewise related, and B - C is more closely related to D than E - F is.

The compiler will automatically provide the necessary coding in the object program to bring the proper segment in memory as required by GO, PERFORM or normal statement sequence. Each segment must always be brought back in its altered form. Thus, if any statement in a procedure is altered, or the value of any variable changed, the compiler must keep track of their altered forms. Each implementor has the option of providing some type of segmentation for working storage, files, etc. In no case does the programmer have any means of indicating segmentation for data.

3. SEQUENCE NUMBERS

In order to facilitate corrections and changes, the concept of sequence numbers has been introduced. A sequence number consists of 6 digits. It corresponds to the line on the coding form prior to the initial compilation, or to the listing on subsequent compilations. Corrections may be made with reference to the sequence numbers. The functions of INSERTION, DELETION and REPLACEMENT of entire lines are desirable. The details will be specified by each implementor.

APPENDIX A

COMPREHENSIVE RULES FOR FORMING ALGEBRAIC EXPRESSIONS

Arithmetic expressions may contain field names, constants, and literals joined by arithmetic operators. Subexpressions may be contained in parentheses as required. The rules for forming arithmetic expressions are as follows:

1. The basic operators are:

OPERATORS	WRITTEN AS
Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	**
Negation	-

2. The ways in which symbol pairs may be formed are summarized in the following table:

		Second Symbol				
		Variable	+-*/**	Negation	(	)
FIRST SYMBOL	Variable	-	P	-	-	P
	+-*/**	P	-	-	P	-
	Negation	P	-	-	P	-
	(	P	-	P	P	-
	)	-	P	-	-	P

Where "P" indicates a permissible symbol pair, and "-" indicates a pair which is not permitted. Thus, "*(" is permissible, while "(*" is not.

3. When the hierarchy of operations in an expression is not completely specified by parentheses, the order of operations (working from inside to outside) is assumed to be exponentiation, then multiplication and division and finally, addition and subtraction. Thus, the expression $A + B/C + D**E*F - G$ will be taken to mean $A + (B/C) + (D^{E \cdot F}) - G$.

4. When the sequence of consecutive operations of the same hierarchal level (i. e., consecutive multiplications and divisions or consecutive additions and subtractions) is not completely specified by parentheses, the order of operations is assumed to be from left to right. Thus, expressions ordinarily considered ambiguous, e.g., A/B*C and A/B/C, are permitted in COBOL statements. For instance, the expression A*B/C*D is taken to mean ((A*B)/C)*D.

5. The expression A^{B^C}, which is sometimes considered meaningful, cannot be written as A**B**C; it should be written as (A**B)**C or A**(B**C), whichever is intended.

APPENDIX B

RULES FOR FORMING COMPOUND CONDITIONS

Conditional expressions may contain conditions, variables, constants, functions, literals, arithmetic operators, relations of equality and relative magnitude and the logical operators NOT, AND and OR. Sub-expressions may be contained in parentheses as required.

If either a condition-name (such as MARRIED) or a relation (such as PAY IS GREATER THAN $2*X+Y$) or a test is designated by the symbol C_i the following rules may be stated concerning the formation of compound conditions involving C_i, NOT, AND and OR.

1. | The Condition | Is True If |
   |---|---|
   | C_1 | C_1 is true |
   | NOT C_1 | C_1 is false |
   | C_1 AND C_2 | Both C_1 and C_2 are true |
   | C_1 OR C_2 | Either C_1 is true, C_2 is true or both are true |
   | NOT (C_1 AND C_2) | "NOT C_1 OR NOT C_2" is true |
   | NOT (C_1 OR C_2) | "NOT C_1 AND NOT C_2" is true |

2. If C_1 and C_2 are compound, then "C_1 AND C_2" and "C_1 OR C_2" are compound conditions, as are similar expressions formed with the use of NOT. Thus, given an expression of the form:

   $$C_1 \text{ AND } (C_2 \text{ OR NOT } (C_3 \text{ OR } C_4))$$

   it may be successively reduced by substituting as follows:

   Let C_5 equal "C_3 OR C_4" resulting in C_1 AND (C_2 OR NOT C_5)

   Let C_6 equal "C_2 OR NOT C_5" resulting in C_1 AND C_6

   Let C_7 equal "C_1 AND C_6" resulting in C_7

   This rule indicates how compound conditions may be formed from simple conditions.

3. The conditional expression "C_1 OR C_2 AND C_3" is identical with "C_1 OR (C_2 AND C_3)" but is not the same as "(C_1 OR C_2) AND C_3". In other words, compound conditions are grouped first according to AND and subsequently by OR. However, the programmer's use of parentheses will affect the order of grouping.

4. The rules for formation of symbol pairs are contained in the following table:

		Second Symbol					
		C	OR	AND	NOT	(	)
First Symbol	C	-	P	P	-	-	P
	OR	P	-	-	P	P	-
	AND	P	-	-	P	P	-
	NOT	P	-	-	-	P	-
	(	P	-	-	P	P	-
	)	-	P	P	-	-	P

where the "P" indicates that the pair is permissible, and the "-" indicates a symbol pair that is not permissible. Thus, the pair "OR NOT" is permissible, while "NOT OR" is not permissible.

APPENDIX C

COMPLETE LIST OF RESERVED WORDS

The words shown below are an inherent part of the COBOL System. Users should avoid choosing these for data or procedure names.

ABOUT	DEFINE	INPUT-OUTPUT
ACCEPT	DEPENDING	INTO
ADD	DISPLAY	IS
ADDRESS	DIVIDE	JUSTIFIED
ADVANCING	DIVIDED	LABEL
AFTER	DIVISION	LEADING
ALL	DOLLAR	LEAVING
ALPHABETIC	ELSE	LEFT
ALPHANUMERIC	END	LESS
ALSO	ENDING	LIBRARY
ALTER	ENDING-FILE-LABEL	LINES
ALTERNATE	ENDING-TAPE-LABEL	LOCATION
AN	END-OF-FILE	LOCK
AND	END-OF-TAPE	LOW-VALUE
APPLY	ENTER	LOW-VALUES
ARE	ENVIRONMENT	MEMORY
AREA	EQUAL	MEMORY-DUMP
AREAS	EQUALS	MEMORY-DUMP-KEY
AS	ERROR	MINUS
ASSIGN	EVERY	MODE
AT	EXACTLY	MODULES
BASE	EXAMINE	MOVE
BASIC	EXCEEDS	MULTIPLE
BEFORE	EXIT	MULTIPLIED
BEGINNING	EXPONENTIATED	MULTIPLY
BEGINNING-FILE-LABEL	FD	NEGATIVE
BEGINNING-TAPE-LABEL	FOR	NO
BITS	FILE	NO-MEMORY-DUMP
BLANK	FILE-CONTROL	NOT
BLOCK	FILLER	NOTE
BLOCK-COUNT	FILLING	NUMERIC
BY	FIRST	OBJECT-COMPUTER
CHARACTERS	FLOAT	OCCURS
CHECK	FORMAT	OF
CLASS	FROM	OFF
CLOCK-UNITS	GIVING	OMITTED
CLOSE	GO	ON
COBOL	GREATER	OPEN
COMPUTE	HASHED	OPTIONAL
CONSTANT	HERE	OR
CONFIGURATION	HIGH-VALUE	OTHER
CONTAINS	HIGH-VALUES	OTHERWISE
CONTROL	I-O-CONTROL	OUTPUT
COPY	IF	PERFORM
CORRESPONDING	IN	PICTURE
DATA	INCLUDE	PLACES
DATE-WRITTEN	INPUT	PLUS

POINT	RIGHT	TEST-PATTERN
POSITION	RUN	THAN
POSITIVE	SAME	THEN
PREPARED	SECTION	THRU
PRIORITY	SELECT	TIMES
PROCEDURE	SENTINEL	TO
PROCEED	SEQUENCED	TYPE
PROTECTION	SIGN	UNEQUAL
PURGE-DATE	SIGNED	UNROUNDED
RANGE	SIZE	UNTIL
READ	SOURCE-COMPUTER	UPON
RECORD	SPACE	USE
RECORD-COUNT	SPACES	VALUE
RECORDING	SPECIAL-NAMES	VARYING
RECORDS	STANDARD	VERB
REDEFINES	STATUS	WHEN
REEL	STOP	WITH
REEL-NUMBER	SUBTRACT	WORDS
RENAMING	SUPERVISOR	WORKING-STORAGE
REPLACING	SUPPRESS	WRITE
RERUN	SYNCHRONIZED	ZERO
RESERVE	TALLY	ZEROES
REVERSED	TALLYING	ZEROS
REWIND	TAPE	